UNIVERSIDAD NACIONAL MAYOR DESAN MARCOS

FACULTAD DE INGENIERÍA ELECTRÓNICA Y ELÉCTRICA

E.A.P. INGENIERÍA ELECTRÓNICA

Control adaptivo de un motor DC paralelo empleando linealización por realimentación

TESIS

para obtener el Título de Ingeniero Electrónico.

AUTOR.

Julio César Tafur Sotelo

ASESOR:

Salomón Luque Gamero

Lima – Perú

2007

A mis padres Herlinda y Marcelino por su aliento y apoyo a lo largo de mi vida.

A Belina, su amor, entrega y confianza son las fuentes inagotables de mi motivación y superación.

AGRADECIMIENTO

Ante todo, quisiera agradecer a mis profesores que me dieron un conocimiento muy valioso y formaron mis valores a lo largo de toda mi carrera.

Mientras estuve estudiando en la Facultad de Ingeniería Electrónica tuve el placer y privilegio de interactuar con muchas personas, agradezco a todos ellos por su apoyo moral.

También me gustaría agradecer en forma especial a mi asesor de tesis, el Ing. Salomón Luque Gamero por su confianza en mi y sus comentarios constructivos.

I N D I C E

LISTA DE FIGURAS

Lista de Tablas

RESUMEN

Esta tesis trabaja en el desarrollo de un Control adaptivo de un motor paralelo empleando linealización por realimentación. Debido al conocimiento impreciso de los parámetros del motor, un enfoque adaptable agrega robustez a la técnica de linealización entrada-salida que requiere la exacta cancelación de los términos no lineales. El objetivo del control es seguir una velocidad de referencia aún cuando se tenga parámetros desconocidos y perturbaciones de carga.

La estabilidad de la dinámica interna es asegurada por el análisis de la dinámica de ceros. Se ha desarrollado también un observador no lineal para el torque de carga con dinámica de error lineal. Las simulaciones se efectuaron usando el programa de simulación SIMULINK™ desde MATLAB™.

El desempeño del controlador adaptivo se ha probado en tiempo real usando una configuración experimental.

Se presenta una metodología para elaborar un programa de control en tiempo real que corra en tiempo real usando El SIMULINK Real-Time Workshop .

La respuesta de velocidad de un motor DC paralelo se forzó a seguir un modelo de referencia de segundo orden con un tiempo de establecimiento de 20 segundos y un tiempo de subida de 10 segundos. Los resultados muestran que la **ley de control adaptivo** efectivamente agrega robustez a la supresión exacta de los términos no lineales y garantiza la convergencia del seguimiento.

ABSTRACT

This thesis work deals with the development of a real-time adaptive feedback linearizing controller for a DC shunt motor. Due to the imprecise knowledge motor parameters, an adaptive approach adds robustness to an input-output linearization technique that requires exact cancellation of nonlinear terms. The control goal is to track a reference speed under unknown parameters and load disturbances. The stability of internals dynamics is assured by analysis of zero dynamics. A nonlinear observer for load torque with linear error dynamics was also developed to give a load torque estimate to the adaptive controller. The simulations were carried out using simulation software SIMULINK™ from MATLAB™.

The performance of the adaptive controller was tested in real-time using a experimental setup. A methodology for building a real-time control program that runs in real time using the SIMULINK Real-Time Workshop was presented.

The speed response of the DC shunt motor was forced to track a second order model reference with settling time of 20 sec. and a rise time of 10 sec. The results show that the adaptive control law effectively adds robustness to the exact cancellation of nonlinear terms and guarantees tracking convergence.

Capítulo 1

Introducción

1.1 Objetivos

Los objetivos de este trabajo de tesis son el diseño e implementación de un controlador adaptivo para un motor DC paralelo empleando linealización por realimentación.

El objetivo del sistema de control es que la velocidad del rotor siga la velocidad referencia bajo parámetros desconocidos y perturbaciones de la carga.

1.2 Justificación

1.2.1 Científicas

El modelo matemático de un motor DC paralelo es no-lineal, y ello complica su uso en las aplicaciones que requieren control automático de la velocidad. Un método tradicional de control de sistemas no lineales es el método de linealización, el cual trata con pequeñas perturbaciones alrededor de los puntos de operación.

Entonces, usando técnicas de diseño de sistema lineales bien conocidos, se puede diseñar un controlador que satisfaga las especificaciones de trabajo alrededor del punto de operación. Este método no será suficiente cuando grandes perturbaciones afecten al sistema. Por lo tanto, para lograr un control de alto rendimiento, es necesario usar técnicas de control no lineal.

Existe un considerable desarrollo en la teoría de control no lineal en los últimos años con enfoques tal como la linealización por realimentación. Sin embargo, la implementación real de los tales algoritmos de control han sido pocos.

Aunque la teoría de los motores DC esta bien establecido (Krause, 1983; Leonhard, 1985; Fitzgerald et, al., 1983), los parámetros del subsistema mecánico podrían ser desconocidos, puesto que estos parámetros están relacionados a la carga del motor, la cual comúnmente esta sujeta a cambios.

También, es razonable considerar la resistencia y la inductancia del modelo de la parte eléctrica del motor, puesto que ellos son desconocidos o varían significativamente con la temperatura y las condiciones de operación.

La técnica de control de linealización por realimentación de sistemas no lineales requiere la cancelación de los términos no lineales. Cuando estos términos contienen parámetros desconocidos, se deben emplear las técnicas de control adaptivas.

Debido a estos problemas, la motivación de esta tesis es considerar el control adaptivo para un motor DC paralelo. Este trabajo se dirige al problema de diseñar e implementar un controlador adaptivo no lineal para un motor DC paralelo usando linealización entrada-salida.

Este es un enfoque de control no lineal que cancela las no linealidades de sistemas por medio de la realimentación de la variable de estado. Las técnicas de diseño lineales pueden aplicarse al sistema linealizado por realimentación.

1.2.2 Económicas

Las aplicaciones de control de movimiento hacen uso extensivo de motores DC debido a su simplicidad relativa en lograr alto rendimiento con control preciso. En general, sus mayores ventajas están en la flexibilidad y versatilidad en sistemas de posicionamiento y regulación de velocidad. La desventaja principal puede ser la inversión inicial y el mantenimiento.

La mayoría de los sistemas de mando DC, incluyendo sus mandos de motores, son estables en la operación, de modo que solamente el comportamiento en estado estable del motor DC

se considera durante el proceso de diseño. Además, para tales sistemas, un motor DC paralelo excitado desde una única fuente es frecuentemente satisfactorio y proporciona un rango ajustable razonable de velocidad y torque.

El motor DC paralelo se usa esencialmente para aplicaciones de velocidad constante que requieren un torque medio de inicio, tales como bombas centrífugas, ventiladores, transportadores, máquinas aserradoras, máquinas herramientas, y prensas de impresión.

1.2.3 Sociales

Sentar las bases para el diseño de sistemas de control de movimiento que promueva el adquirir conocimiento y técnicas aplicadas al control de movimiento para solución a problemas en el campo de la automatización de producción y fabricación con la finalidad de optimizar estos procesos.

1.3 Motivación

La respuesta entrada-salida de algún sistema no lineal puede volverse lineal por medio de realimentación de estado y transformaciones de coordenadas. La teoría esta ahora bien desarrollada y entendida (Jakubcyk y Respondek, 1980; Hunt et al., 1983; Isidori y Ruberti, 1984; Baumann y Rugh, 1986; Zeitz, 1987; Charlet y Lévine, 1989; Hirschorn, 1990; Isidori, 1989; Slotine y Li, 1991).

La aplicabilidad de linealización por realimentación al motor DC paralelo fue investigado en (Oliver, 1991) y en (Bodson y Chiasson, 1993). Oliver discute la linealización por realimentación por entrada-estado y entrada-salida de motores DC series y paralelos, y determina las condiciones bajo las cuales la dinámica de estos motores son linealizables por realimentación. Bodson y Chiasson presenta el problema de controlar un motor DC paralelo, en el cual diversos métodos para control no-lineal son comparados, incluyendo la linealización por entrada-estado, formas canónicas generalizadas, y la linealización de la entrada-salida.

En (Chiasson y Bodson, 1991) se trata el problema de diseño de un observador de torque no lineal con dinámica de error lineal, y da ideas para diseñar un controlador adaptivo para un motor DC paralelo, pero no muestra resultados basados en la aplicación de este algoritmo adaptivo. También en (Chiasson, 1994) el problema de controlar un motor DC serie es considerado. Puesto que el modelo matemático de este motor es no-lineal, dos leyes de realimentación no lineal se consideran basados en la linealización de entrada-estado y la linealización de la entrada-salida. También un observador no-lineal con la dinámica de error lineal se construye para el motor DC serie que hace la estimación de velocidad y torque de carga basado en la medición de corrientes (corrientes de armadura y de campo). El desempeño del algoritmo observador/controlador propuesto es estudiado por simulaciones. Aunque algunos trabajos relacionados al control de motor DC paralelo por linealización por realimentación cercanos han sido reportados anteriormente, estos carecen de resultados experimentales.

1.4 Aportes

- La contribución de esta tesis radica en el área de control de movimiento. En esta área se contribuye con un proyecto de Control Adaptivo estable para una linealización adaptiva de las no linealidades de un motor DC paralelo.
- Otra contribución es la demostración de la factibilidad de implementar tales algoritmos no lineales de control en tiempo real usando SIMULINK Real-Time Workshop™.

1.5 Estructura de la Presentación

Esta tesis se organiza como se indica a continuación. En el Capítulo 2, el modelo matemático de un motor DC paralelo es presentado. En el Capítulo 3, los antecedentes teóricos para la aproximación de linealización por realimentación es presentada y el controlador de velocidad linealizando entrada-salida se plantea dentro de un marco de un formato estándar para los sistemas linealizables por realimentación tipo una entrada-una salida (*Single-Input – Single Output*, SISO).

En el Capítulo 4, la teoría desarrollada en (Sastry y Bodson, 1989), (Sastry e Isidori, 1989) para el control adaptivo del sistema linealizable está presente y se muestra los pasos a seguir para desarrollar un seguimiento de velocidad adaptiva para un motor DC paralelo basado en modelo referencia. También, en este capítulo, un observador no-lineal para el torque de carga con error dinámico linealizable es presentado.

En el Capítulo 5, las simulaciones del algoritmo adaptivo que se desarrollaron en el Capítulo 4 para un control de velocidad de motor DC paralelo son probadas bajo las perturbaciones de carga y parámetros desconocidos del sistema.

El capítulo 6 presenta la metodología para implementar un algoritmo de linealización adaptivo. El código ha desarrollarse se implementara usando el programa SIMULINK Real-Time Workshop™.

Finalmente se describirán las conclusiones y las recomendaciones.

Capítulo 2

Modelamiento del Motor DC Paralelo

Este capítulo discute el modelo matemático de motor DC paralelo. Un modelo matemático de este motor se define como un conjunto de ecuaciones diferenciales no-lineales que representan la dinámica del motor DC paralelo. Usando la notación matricial, una ecuación diferencial de segundo orden se expresa como una representación de espacio de estado.

También, en este capítulo, son descritos el procedimiento para estimar el valor nominal de los parámetros que se usan para describir la dinámica del sistema.

2.1. Modelo del Motor

Un motor DC paralelo es un motor DC donde el bobinado de campo se conecta en paralelo con el bobinado de armadura y con la salida de la fuente de alimentación. Los valores nominales del motor se muestran en la Tabla 1.

Tabla 1: Valores Nominales del Motor DC Bobinado

	Valores Nominales
Potencia	125 Watts
Voltaje	125 Voltios
Velocidad de Rotor	1725 RPM
Corriente de Armadura	1.0 Amperios
Corriente de Campo	0.44 Amperios

Un diagrama esquemático de este motor se muestra en la Figura 1.

Aquí asumimos el flujo *direct-air-gap* es linealmente proporcional a la corriente de campo i_F (Krause, 1983; Leonhard, 1985; Fitzgerald et al., 1985). La ecuación diferencial para el circuito de campo es

$$L_F \frac{di_F}{dt} + (R_{adj} + R_F)\, i_F = V_T \qquad (1)$$

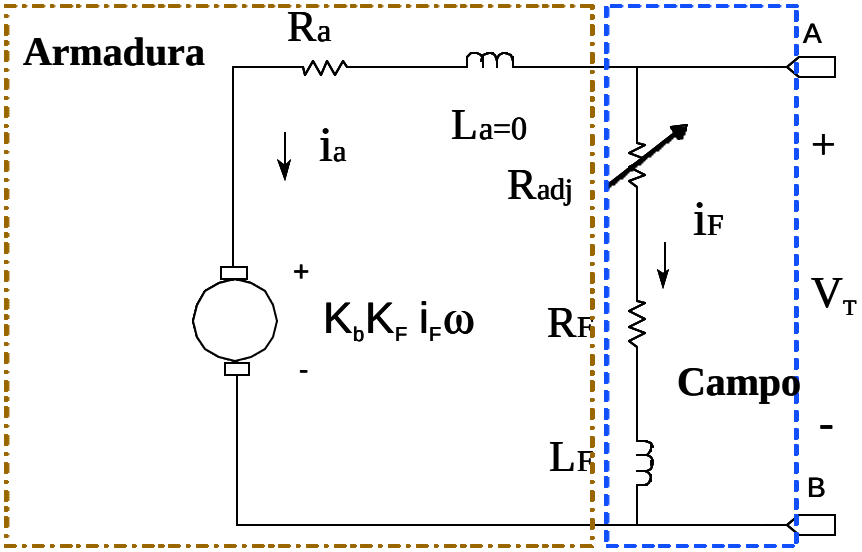


Figura 1: Dia grama Esquemático del motor DC paralelo

La corriente de armadura y de campo produce un torque que se aplica a la inercia, fricción y la carga conectada al eje del motor; de aquí en adelante

$$J\frac{dw}{dt} + Bw + t_L = t_m \qquad (2)$$

La ecuación diferencial para la circuito de armadura es

$$L_a \frac{di_a}{dt} + R_a\, i_a + K_b\, K_F\, i_F\, w = V_T \qquad (3)$$

donde:

$t_m = K_m\, f\, i_a$; Torque electromagnético

K_m ; Constante de torque

$f = K_F i_F$; Flujo de armadura debido al campo

K_F ; Inductancia mutua

i_a ; Corriente de armadura

i_F ; Corriente de campo

$R_{adj} + R_F$; Resistencia de campo (R_{adj} es variable)

t_L ; Torque de carga

L_a ; Inductancia de la armadura

L_F ; Inductancia del campo

J ; Momento de inercia del rotor

B ; Coeficiente de fricción viscosa

ω ; Velocidad angular

V_T ; Voltaje terminal

Una suposición usual es que la inductancia de la armadura es insignificante y aproximadamente cero (en nuestro caso $L_a = 23.472mH$ y $L_F = 24.736H$) comparada con la inductancia del campo.

Usando (3) la corriente de armadura es dado por

$$i_a = \frac{V_T - V_b}{R_a},$$

donde $V_b = K_b K_F i_F w$ es el voltaje back-emf, R_a es la resistencia de armadura y K_b es la constante back-emf. Usando las unidades MKSA, puede mostrarse por el argumento de conservación de energía que $K_m = K_b$.

2.1.1 Modelo de Descripción en espacio de Estado

Un modelo de espacio de estado para el sistema de motor DC paralelo puede obtenerse como se indica a continuación. Se define las variables de estado x_1 y x_2 por

$$x_1 = w$$

$$x_2 = i_F$$

la variable de entrada u por

$$u = V_T$$

y la variable de salida y por

$$y = \omega = x_1$$

Entonces la representación del espacio de estado de un sistema de motor DC paralelo esta dado por:

$$\begin{aligned} \dot{x}_1 &= -c_1 x_1 - c_2 x_1 x_2^2 + c_3 x_2 u - c_6 t_L \\ \dot{x}_2 &= -c_4 x_2 + c_5 u \\ y &= x_1 \end{aligned} \tag{4}$$

donde:

$$c_1 = \frac{B}{J}$$

$$c_2 = \frac{K_m K_b K_F^2}{J R_a}$$

$$c_3 = \frac{K_m K_F}{J R_a}$$

$$c_4 = \frac{R_{adj} + R_F}{L_F}$$

$$c_5 = \frac{1}{L_F}$$

$$c_6 = \frac{1}{J}$$

Reescribimos (4) en la forma estándar para ser usado luego en la formulación de La ecuación del modelo linealizado entrada-salida.

La forma estándar esta dada por:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u + pt_L \\ y &= x_1 \end{aligned} \tag{5}$$

donde:

$$f(x) = \begin{bmatrix} -c_1 x_1 - c_2 x_1 x_2^2 \\ -c_4 x_2 \end{bmatrix}, g(x) = \begin{bmatrix} c_3 x_2 \\ c_5 \end{bmatrix}, p = \begin{bmatrix} -c_6 \\ 0 \end{bmatrix}$$

2.2 Estimación de Parámetros.

En esta sección, los métodos experimentales se ponen de manifiesto para la determinación de los parámetros del modelo motor DC presentado en la última sección.

- Las resistencias, R_a y R_F, son medidas por el método del voltimetro-amperimetro (McPherson, 1990).

- El K_b del back-emf es determinado por medición en el voltaje terminal de la armadura en circuito abierto cuando el motor es operado como un generador DC.

$$V_a = K_b K_F i_F w$$

entonces:

$$K_b K_F = \frac{V_a}{i_F w} \tag{6}$$

Notamos que para el modelo del motor DC paralelo, nosotros necesitamos solo conocer el producto $K_b K_F$, asumiendo que este producto sea constante en el rango de operación.

Los valores siguientes se midieron del circuito mostrado en la Figura 1:

$Ra = 15.9\Omega$, $R_F = 267\Omega$, $\omega = 112.89 rad/sec$, $i_F = 0.195A$ y $Va = 59.1 volts$.

El valor de $K_b K_F = 25.6848 H$ es obtenido reemplazando los valores anteriores (6)

- La inductancia propia L_a y L_F son medidas por el método transitorio (Venkatesan y Mukhopadhyay, 1972).

El voltaje a través de un resistor en serie con el bobinado, el cual es proporcional a la respuesta de corriente que se registra en un osciloscopio.

La inductancia propia se calculó desde la medición de la constante de tiempo (t). Asumiendo que el tiempo de establecimiento, t_s y la constante de tiempo , t son relacionados por $t_s = 4t$ entonces la inductancia propia L es dada por

$$L = R_e \frac{t_s}{4},$$

donde R_e es el equivalente de resistor del circuito L - R.

Para el circuito de armadura $t_s = 0.1448 ms$, $R_e = 517.9\Omega$ entonces $La = 18.745 mH$.

Para el circuito de campo $ts = 128 ms$, $Re = 773\Omega$, entonces $L_F = 24.736 H$.

- El coeficiente de fricción puede ser determinado por usar (2) en el estado

estable $\left(\frac{dw}{dt} = 0, \frac{di_F}{dt} = 0 \right)$ y tener $t_L = 0$. puesto que

$$\tau_m = K_m K_F i_F i_a = K_m K_F I_F I_a$$

y puesto que $\frac{dw}{dt} = 0$ desde (2), nosotros obtenemos

$$K_m \; K_F \; I_F \; I_a \; = Bw$$

esto resulta en

$$B = \frac{K_m \; K_F \; I_F \; I_a}{w}$$

para $I_F = 0.195A$, $Ia = 0.45A$, $w = 108.385 rad/seg$, nosotros obtenemos $B = 0.0022 N\text{-}m/rad/sec$ aproximadamente.

- El momento de inercia del rotor *(J)* es determinada por las características del retardo de la velocidad de prueba versus el tiempo cuando el motor es apagado después que haya alcanzado el estado estable (Venkatesan y Mukhopadhyay, 1972).

Usando la ecuacion (2) y considerando $\tau_m = 0$ y $\tau_L = 0$ (sin carga mecánica), el resultado es:

$$J \frac{dw\ (t)}{dt} + Bw\ (t) = 0$$

y la solución de la ecuación diferencial lineal da

$$w\ (t) = w\ (0) \exp\left(-\frac{B}{J} t \right) = w\ (0) \exp\left(-\frac{t}{t} \right),$$

donde $\tau = J/B$ es la constante de tiempo mecánica del motor.

Las curvas w versus tiempo se registró en una PC y se muestra en la Figura 2.

De los datos registrados, $w(0) = 154.058 rad/seg$; $\omega = 101.7716 rad/seg$, resulta en $J = 0.01 N\text{-}m/rad/seg$ aproximadamente.

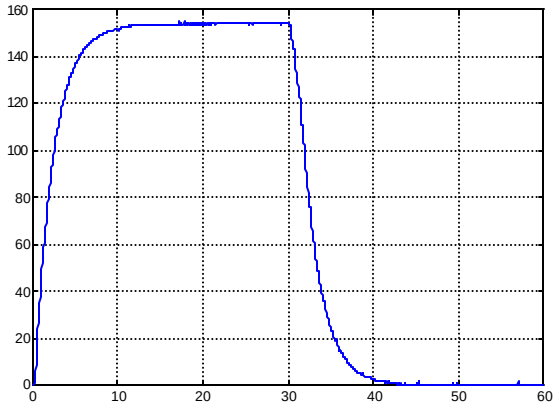


Figura 2: Respuesta de Motor DC paralelo en lazo abierto

Los parámetros del motor bobinado DC se dan en la Tabla 2.

Tabla 2: Parámetros del motor DC.

Parametro	**Valores**
R_a	15.9Ω
R_F	167Ω
L_a	$18.745mH$
L_F	$24.736H$
$K_b K_F$	$2.5H$
J	$0.01N - m/rad/seg^2$
B	$0.0022N - m/rad/seg$

Capítulo 3

Linealización por Realimentación de entrada-salida SISO

En este capítulo se revisa los conocimientos teóricos para una linealización por realimentación de una-entrada una-salida (SISO), y se presenta su aplicación al motor DC paralelo. La teoría de linealización por realimentación es bien conocida y entendida, como puede verse por ejemplo, en los artículos que se han expuesto en (Isidori, 1989) para el caso de tiempo continuo.

El propósito del método consiste en encontrar una transformación no-lineal de espacio de estado tal que en las nuevas coordenadas y por realimentación de estado, el sistema no-lineal se transforma en uno total o parcialmente lineal, esto es, un sistema con parte lineal y parte no-lineal. La linealización por realimentación se ha usado exitosamente para resolver problemas de control prácticos, estos incluyen el control de una aeronave de gran performance, robots industriales, y dispositivos biomédicos.

3.1 Introducción.

Para un sistema SISO, existen dos condiciones que garantizan que la linealización por realimentación sea posible, por lo menos matemáticamente. Esta son la condición de controlabilidad y la condición de involutividad. Antes de presentar estas condiciones, se requiere introducir las definiciones y las notaciones siguientes.

Definición 1 *El vector de función f: R^n ® R^n es llamado vector de campo en R^n. Un vector de campo es uniforme si la función f (x) tiene derivada parcial continua de cualquier orden. (Slotine y Li, 1991)*

Definición 2 *El diferencial, o gradiente, Ñ h de una función escalar de valor-real h: R^n ® R es dado por*

$$\nabla h(x) = dh(x) = \frac{\partial h}{\partial x}$$

$$\nabla h(x) = dh(x) = \left[\frac{\partial h}{\partial x_1} \quad \frac{\partial h}{\partial x_2} \quad \cdots \quad \frac{\partial h}{\partial x_n} \right]$$

$\forall x \in U$, donde U es un subconjunto abierto de R^n.

Definición 3 *Para una función diferenciable f: $R^n \to R^n$, la matriz Jacobiana ($\tilde{N}_f$) definida sobre un subconjunto abierto U de R^n es una matriz m × n de elementos* $(\nabla f)_{ij} = \frac{\partial f_i}{\partial x_j}$

3.1.1 Derivada Lie y Paréntesis Lie.

Dada una función escalar h (x) y un vector de campo f (x), definimos una nueva función escalar $L_f h$, llamada derivada Lie de h con el respecto a f. (Slotine y Li, 1991)

Definición 4 *Considere $h : R^n \to R$ una función escalar uniforme, y $f : R^n \to R^n$ un vector de campo suave en R^n, entonces la derivada Lie de h con respecto a f es una función escalar definida por*

$$L_f h = \nabla h f$$

En una forma recursiva

$$L_f^i h = L_f \left(L_f^{i-1} h \right) = \nabla \left(L_f^{i-1} h \right) f \qquad \text{para } i = 1,2,...$$

donde $L_f^0 h = h$

En forma similar, si g es otro vector de campo, entonces la función escalar

$L_g L_f h(x)$ es

$$L_g\ L_f\ h = \nabla\left(L_f\ h\right)g$$

Definición 5 *Considere f y g dos vectores de campo en R^n. La adjunta (corchetes Lie) de f y g es un tercer vector de campo definido por*

$$ad_f\ g = [f, g] = \nabla g\ f - \nabla f\ g$$

En forma recursiva

$$ad_f^{\,i}\ g = \left[f, ad_f^{\,i-1}\ g\right] \qquad \text{para } i = 1, 2,...$$

donde $ad_f^0 g = g$

Lema 1 *La adjunta (corchetes Lie) tiene las siguientes propiedades*

(i) Bilinealidad:

$$[a_1 f_1 + a_2 f_2, g] = a_1[f_1, g] + a_2[f_2, g]$$

$$[f, a_1 g_1 + a_2 g_2] = a_1[f, g_1] + a_2[f, g_2]$$

donde f, f_1, f_2, g, g_1, y g_2 son vectores de campo suaves, y a_1 y a_2 son constantes escalares.

(ii) Commutatividad:

$$[f, g] = -[g, f]$$

(iii) Identidad de Jacoby;

$$L_{ad_{fg}} h = L_f\ L_g\ h - L_g\ L_f\ h$$

donde $h(x)$ es una función escalar suave en x.

3.1.2 Difeomorfismo.

El concepto de difeomorfismo es una generalización del concepto de transformación de coordenadas. Formalmente se define como se indica a continuación:

Definición 6 *Una función* $f : R^n \to R^n$, *definida en una región* W, *se llama un difeomorfismo suave, si su inversa* f^{-1} *existe y es suave.*

El lema siguiente nos permite verificar si la función no lineal $\phi(x)$ es un difeomorfismo local.

Lema 2 *Considere que* $f(x)$ *es una función suave definida en una región* W *en* R^n. *Si la matriz Jacobiana* $\tilde{N}f$ *es no singular en un punto* $x = x_0$ *de* W, *entonces* $f(x)$ *define un difeomorfismo local en una subregion de* W. *(Slotine y Li, 1991)*

Este difeomorfismo se puede usar para transformar un sistema no-lineal dentro de otro sistema no-lineal en términos de un nuevo conjunto de coordenadas.

3.1.3 Condiciones para la Linealización por Realimentación de Sistemas No lineales SISO.

Considere el sistema no lineal SISO

$$x = f(x) + g(x)\,u \qquad (7)$$

con $x \hat{I} R^n$; $f, g,$ suave.

Definición 7 *Supone n, m son enteros dados,* $A \hat{I} R^{n \times n}$ *y* $B \hat{I} R^{n \times m}$. *Entonces el par (A, B) se dice que es controlable si*

$$rango\left[B \;\; AB \;\; \ldots \;\; A^{n-1}B\right] = n$$

Definición 8 *Un conjunto de vectores es declarada involutivo si*

$$rango\{g_1,...,g_k\}=rango\{g_1,...,g_{k-1},ad_{g_i}g_j\} \quad \forall \quad i,j\in\{1,...,k\}$$

Esto es, añadiendo los corchetes de Lie de cualquiera de dos miembros del sistema resulta en el rango de los vectores del sistema. Los lectores interesados pueden referirse a (Sbtine y Li, 1991) para una introducción mas completa.

Si esta condición no es satisfecha, no hay posibilidad que una transformación pueda existir para un sistema no-lineal y así es de gran interés encontrar condiciones que aseguran la linealización por realimentación.

(i) Condición de Controlabilidad. La condición de controlabilidad requiere que para un sistema de orden n, la matriz

$$C=\left[g \quad ad_f\, g \ \ldots \ ad_f^{n-1}\, g\right]$$

Sea no singular.

La primera condición puede interpretarse como una representación simple de la condición de controlabilidad para el sistema no lineal (7), ello simplemente asegura que dado un sistema Ω que contiene el origen y sobre la cual la transformación ϕ (x) es un difeomorfismo, el sistema puede llevarse desde un punto arbitrario en Ω al origen.

Comentario 1 *Considere que f (x) tiene un punto de equilibrio en x = 0, y considere para f (x) una expansión de la forma*

$$f(x)=Ax+f_2(x)$$

con

$$A=\left[\frac{\partial f}{\partial x}\right]_{x=0} \quad y \quad \left[\frac{\partial f_2}{\partial x}\right]_{x=0}=0$$

el cual separa al sistema en una aproximación lineal Ax del término de mayor f_2 (x). Considere también para g (x) una expansión de la forma

$$g(x) = B + g_1(x)$$

con B = g(0). La aproximación lineal del sistema a x = 0, es definido como

$$\dot{x} = Ax + Bu$$

Después de algunos cálculos los vectores de campo $ad_f^k\, g(x)$ *pueden expandirse de la manera siguiente*

$$ad_f^k\, g(x) = (-1)^k\, A^k\, B + p_k(x)$$

donde $p_k(x)$ *es una función tal que* $p_k(0) = 0$. *La expansión en cuestión es trivialmente cierta para k = 0. Por inducción, se supone que es cierto para algun k.*

Entonces, por definición

$$ad_f^{k+1} g(x) = \frac{\partial \left(ad_f^k g(x)\right)}{\partial} f(x) - \frac{\partial f}{\partial x} ad_f^k g(x)$$
$$= \frac{\partial p_k}{\partial x}\left(Ax + f_2(x)\right) - \left(A + \frac{\partial f_2}{\partial x}\right)\left((-1)^k A^k B + p_k(x)\right)$$
$$= (-1)^{k+1} A^{k+1} B + p_{k+1}(x)$$

donde $p_{k+1}(x)$, *por construcción, es cero a x = 0.*

Desde esto podemos ver que la condición (i) es equivalente la condición de controlabilidad

$$rango\left[B \;\; AB \;\; \ldots \;\; A^{n-1} B\right] = n$$

de una aproximación lineal del sistema en x = 0.

(ii) Condición Involutiva. La condición de involutividad asegura la existencia del difeomorfismo ϕ (x). Ello requiere que el conjunto de funciones vectoriales

$$\{g\ ad_f\ g ... ad_f^{n-2} g\}$$

sea involutiva.

3.1.4 Grado Relativo.

Considere el sistema no lineal de una-entrada una-salida (SISO)

$$\begin{aligned} \dot{x} &= f(x) + g(x)\, u \\ y &= h(x) \end{aligned} \qquad (8)$$

con $x \in R^n$; f, g, h vectores de campo suaves. El sistema SISO (8) se dice que tiene grado relativo r si

$$(i) \quad L_g\, L_f^i\, h(x) = 0 \qquad \forall i < r-1$$

$$(ii) \quad L_g\, L_f^{r-1} h(x) \neq 0$$

Derivando y con respecto al tiempo, obtenemos

$$\dot{y}(t) = L_f\, h(x) + L_g\, h(x)\, u.$$

Si el grado relativo es más mayor que 1, tenemos $L_g h(x) = 0$ y por lo tanto

$$\dot{y}(t) = L_f\, h(x),$$

lo cual resulta

$$\ddot{y}(t) = \frac{\partial\, L_f\, h(x)}{\partial\, x} \frac{dx}{dt} = L_f^2\, h(x) + L_g\, L_f\, h(x)\, u.$$

Nuevamente, si el grado relativo es mayor que 2, nosotros tenemos $L_g\, L_f h(x) = 0$ y

$$\ddot{y}(t) = L_f^2 h(x).$$

Más generalmente,

$$y^{(i)}(t) = L_f^k h(x) \quad \forall i < r$$
$$y^{(r)}(t) = L_f^r h(x) + L_g L_f^{r-1} h(x) u.$$

Así, el grado relativo *r* es exactamente igual al número de veces que se tiene que diferenciar la salida *y(t)* a fin de que el valor de entrada aparezca explícitamente.

Notamos también que si

$$L_g L_f^i h(x) = 0 \qquad \forall \ i \geq 0,$$

Entonces la salida del sistema no es afectada por la entrada.

La teoría es considerablemente más complicada si $L_g L_f^{r-1} h(x) = 0$ para algún valor de x. En esta tesis no se discute este ese caso, pero el lector puede referirse a (Slotine y Li, 1991) para mayores detalles.

Comentario 2 *El concepto de grado relativo es bastante consistente con el caso lineal. Para mostrar esto considere un sistema de una-entrada, una-salida de la forma*

$$\dot{x} = Ax + Bu.$$
$$y = Cx.$$

Puesto que $f(x) = Ax$, $g(x) = B$, $h(x) = Cx$, *es fácil ver que*

$$L_f^k h(x) = C A^k x,$$

y por lo tanto

$$L_g L_f^k h(x) = C A^k B,$$

Así, el entero r es caracterizado por las condiciones

$$C A^k B = 0 \quad \text{para todo } k < r - 1$$
$$C A^{r-1} B \neq 0$$

El entero que satisface estas condiciones es exactamente igual a la diferencia entre el grado del polinomio denominador y el grado del polinomio numerador de la función de transferencia

$$H(s) = C(sI - A)^{-1} B$$

del sistema.

3.1.5 Transformación coordenada.

Suponga que el sistema tiene un grado relativo r. Entonces $r \leq n$. ponemos

$$f_1(x) = h(x)$$
$$f_2(x) = L_f h(x)$$
$$\cdots\cdots$$
$$f_r(x) = L_f^{r-1} h(x)$$

Si r es estrictamente menor que n, es siempre posible encontrar $n - r$ funciones $f_{r+1}(x)$, ... , $f_n(x)$, tal que el mapeo

$$f(x) = \begin{bmatrix} f_1(x) \\ \cdots \\ f_n(x) \end{bmatrix} \qquad (9)$$

tiene una matriz Jacobiana que es no singular y por lo tanto califica como una transformación de coordenada local. Además, es siempre es posible elegir $f_{r+1}(x), \ldots\ldots, f_n(x)$ de tal modo que

$$L_g f_i(x) = 0 \qquad \forall \quad r+1 \le i \le n$$

Ahora, si el mapeo (9) existe, el sistema puede expresarse en el así llamado **forma normal** como sigue

$$\begin{aligned}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\ldots\ldots \\
\dot{z}_{r-1} &= z_r \\
\dot{z}_r &= f(z) + g(z)\,u \qquad (10) \\
\dot{z}_{r+1} &= q(z) \\
&\ldots\ldots \\
\dot{z}_n &= q_n(z)
\end{aligned}$$

donde $y = h(x) = z_1$

La estructura de las ecuaciones es mejor ilustrada en el diagrama de bloques mostrado en la Figura 3

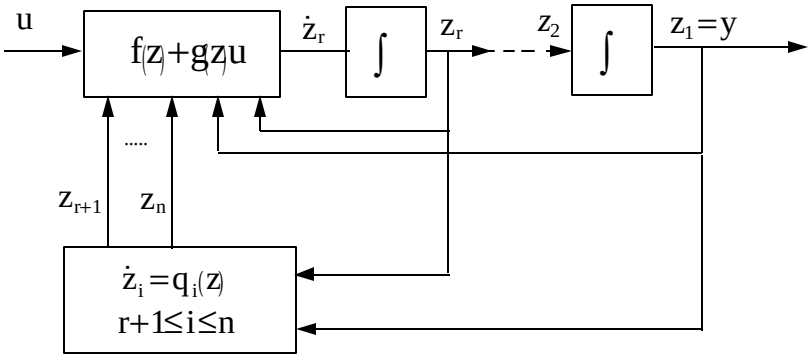


Figura 3: Diagrama Esquemático del Sistema Linealizado.

3.1.6 Dinámica de Cero.

Por medio de la linealización entrada-salida, la dinámica del sistema no-lineal se descompone en una parte externa (entrada-salida) y una parte interna parte (no observable).

Puesto que la dinámica externa puede ser estabilizada por medio de la realimentación, la pregunta de estabilidad se reduce en asegurar la estabilidad de la dinámica interna.

Debido a que el diseño del control debe ser considerado para toda la dinámica, el comportamiento interno debe considerarse cuidadosamente.

Considere un sistema no lineal con grado relativo r estrictamente menor que n y observe su forma normal. Definiendo

$$h \triangleq \begin{bmatrix} z_{r+1} \\ \cdots \\ z_r \end{bmatrix}$$

$$z \triangleq \begin{bmatrix} z_1 \\ \cdots \\ z_r \end{bmatrix}$$

Usando la notación anterior, la forma normal (10) para un sistema no lineal de una-entrada una-salida que tiene $r < n$ se puede escribir como

$$\begin{aligned} \dot{z} &= f(z,n) + g(z,h)u \\ \dot{h} &= q(z,h), \end{aligned} \tag{11}$$

donde las últimas n-r ecuaciones $\dot{\eta} = q(z, \eta)$, de la forma normal constituye la dinámica no observable.

La **dinámica de cero** es la dinámica de η cuando la entrada de control es tal que la salida y se mantiene a cero. Estudiando la dinámica de cero nos permite hacer algunas conclusiones acerca de la estabilidad de la dinámica interna.

Cuando la salida del sistema es idénticamente cero, su estado es forzado a evolucionar de tal modo que también $z(t)$ es idénticamente cero, entonces el comportamiento de $\eta(t)$ se rige por la ecuación diferencial

$$\dot{\eta}(t) = q(0, \eta(t))$$

Si la salida $y(t)$ tiene que ser cero, entonces necesariamente el estado inicial del sistema deberá ser del conjunto de valores tal que $z(0) = 0$, considerando $\eta(0) = \eta_0$ puede ser elegido arbitrariamente. De acuerdo al valor de η_0 de (11), la entrada es ajustada como

$$u(t) = -\frac{f(0, \eta)}{g(0, \eta)} = -\frac{L_f^r \, \eta(f^{1}(0))}{L_g \, L_f^{r-1} \, \eta(f^{-1}(0))}, \qquad (12)$$

donde $\eta(t)$ denota la solución de la ecuación diferencial

$$\dot{\eta}(t) = q(0, \eta(t)), \quad \text{con la condición inicial } \eta(0) = \eta_0$$

Si la dinámica interna es inestable, implica que un esfuerzo de control infinito se requiere para mantener la linealidad de la dinámica z. Si la dinámica de cero es estable, entonces el sistema transformado es estable localmente.

Extensiones pueden realizarse en el problema de seguimiento, ver (Isidori, 1989).

Sin embargo, a diferencia de caso lineal, no se pueden dar resultados sobre la estabilidad global para sistemas dinámicos no lineales.

Similarmente al caso lineal, un sistema no lineal cuyo dinámica de cero es asintoticamente estable se llama un **sistema asintótico de fase mínima.** (Sastry y Bodson, 1989).

La dinámica de ceros es una característica intrínseca del sistema no lineal, que no depende de la elección de la ley de control o las trayectorias deseadas.

Examinar la dinámica de cero es mucho más fácil que examinar la estabilidad de la dinámica interna, debido a que la dinámica de cero solo involucra los estados internos, mientras la dinámica interna se acopla a la dinámica externa y a las trayectorias deseadas.

3.2 Transformación del modelo del sistema Motor DC paralelo.

En esta sección encontramos una transformación no lineal de espacio-estado- tal que en las nuevas coordenadas y por la realimentación de estado, la dinámica del sistema no lineal es transformada en una total o parcialmente lineal.

Recuerde que la descripción del sistema en espacio-estado dado en (5), está en la forma siguiente;

$$\begin{aligned} \dot{x} &= f(x) + g(x)u + pt_L \\ y &= x_1, \end{aligned} \tag{13}$$

con

$$f(x) = \begin{bmatrix} -c_1 x_1 - c_2 x_1 x_2^2 \\ -c_4 x_2 \end{bmatrix}, \; g(x) = \begin{bmatrix} c_3 x_2 \\ c_5 \end{bmatrix}, \; p = \begin{bmatrix} -c_6 \\ 0 \end{bmatrix}$$

donde $x_1 = w$ y $x_2 = i_F$.

La primera derivada de la salida es dado por

$$\dot{y} = L_{f+gu+pt_L}\, h = L_f\, h + L_g\, hu + L_{pt_L}\, h.$$

Puesto que

$$L_g\, h = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} c_3\, x_2 \\ c_5 \end{bmatrix} = c_3\, x_2\,,$$

que es diferente de cero para $x_2 \neq 0\ (i_F \neq 0)$, entonces el grado relativo del sistema es $r = 1$.

3.2.1 Condiciones para la linealización entrada-salida.

Consideramos ahora la transformación de estado

$$f\ (x) = \begin{bmatrix} z_1 \\ n \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix},$$

donde $z_1 = x_1$ y $z_2 = h(x)$ la segunda función requerida para completar la transformación, esta función debería satisfacer

$$L_g h(x) = \frac{\partial h(x)}{\partial x} g_1(x) + \frac{\partial h(x)}{\partial x} g_2(x) = 0$$

$$0 = \frac{\partial h(x)}{\partial x} c_3 x_2 + \frac{\partial h(x)}{\partial x} c_5$$

la solución de esta ecuación es

$$h(x) = -c_5\, x_1 + \frac{c_3}{2} x_2^2$$

entonces

$$f\ (x) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ -c_5\, x_1 + \frac{c_3}{2}\, x_2^2 \end{bmatrix} \qquad (14)$$

La matriz Jacobiana es

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 1 & 0 \\ -c_5 & c_3 x_2 \end{bmatrix},$$

que es no singular para $x_2 \neq 0$ ($i_F \neq 0$). La transformación inversa entonces existe y esta dada por

$$x_1 = z_1$$

$$x_2 = \sqrt{\frac{2}{c_3}(z_2 + c_5 z_1)}$$

Entonces $f(x)$ define un difeomorfismo local en una subregión de Ω. Este dimorfismo puede usarse para transformar un sistema no lineal en otro sistema no lineal en términos de un nuevo conjunto de estados.

3.2.2 Linealización Entrada-Salida del Modelo SISO.

En resumen, la descripción espacio-estado del sistema en las nuevas coordenadas usando la transformación dada en (14), será como se indica a continuación

$$\begin{aligned} \dot{z}_1 &= f(z_1, z_2) + g(z_1, z_2)u \\ \dot{z}_2 &= h(z_1, z_2) = c_1 c_5 z_1 + (c_2 c_5 z_1 - c_3 c_4)(z_2 + c_5 z_1)\frac{2}{c_3} + c_5 c_6 t_L, \end{aligned} \tag{15}$$

donde

$$f(z_1, z_2) = -c_1 z_1 - c_2 z_1 (z_2 + c_5 z_1)\frac{2}{c_3} - c_6 t_L$$

$$g(z_1, z_2) = \sqrt{\frac{2}{c_3}(z_2 + c_5 z_1)}.$$

Usando la ley de control

(16)

El sistema linealizado por realimentación sera como se indica a continuación

$$\dot{z}_1 = v.$$

Donde v es la entrada del lazo linealizado por realimentación.

En las coordenadas originales

La ley de control es

(17)

entonces

$$\dot{y} = v.$$

El sistema original es de segundo orden mientras que el sistema linealizado entrada-salida es de primer orden. Hemos ocultado (hecho no observable) el segundo estado $x_2 = i_F$ por esta ley de realimentación. La ley de control mencionada arriba requiere que x_2 este acotada lejos de cero, como x_2 es la corriente de campo, esta es una condición razonable. La Figura 4 muestra el diagrama de bloques del modelo linealizado del motor DC paralelo con la velocidad como salida.

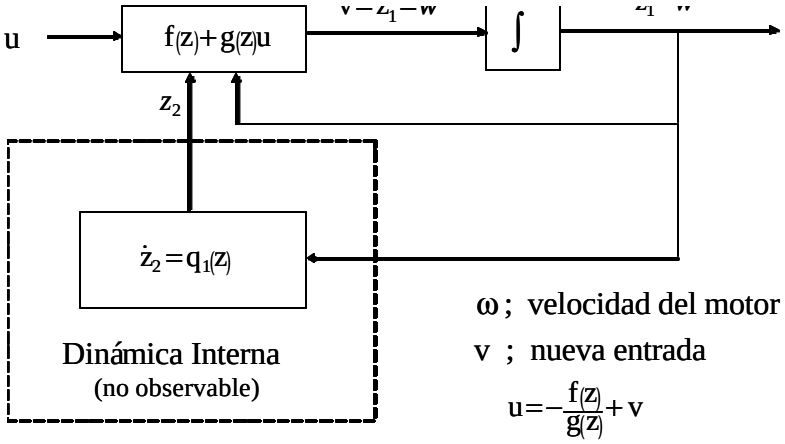


Figura 4: Modelo Linealizado

3.2.3 Estabilidad de la Dinámica de Cero.

La dinámica de cero para (13) son obtenidas haciendo $z_1 = 0$ en $h(z_1, z_2)$ de (15), entonces

$$h(0, z_2) = -2 c_4 z_2 + c_5 c_6 t_L,$$

o

$$\dot{z}_2 + 2c_4 z_2 - c_5 c_6 t_L = 0. \qquad (18)$$

Para $t_L = 0$;

Como [ ] la dinámica de cero es asintoticamente estable, entonces esto es **exponencialmente estable**. (Slotine y Li, 1991)

Para $t_L \neq 0$, consideremos la función candidata a Lyapunov como el cuadrado de la distancia al origen

[ ]

la función V es radialmente no acotada, puesto que tiende al infinito como [ ]

Su derivada

[ ]

donde [ ]

Puesto que [ ] si t_L es acotada, entonces $c_4 > c_5 c_6 t_L$

Así $\dot{V}(z_2) < 0$ en tanto que $z_2 \neq 0$

De modo que $z_2 = 0$ es un **punto de equilibrio globalmente asintoticamente estable** y entonces la dinámica de ceros es **globalmente exponencialmente estable.** (Slotine y Li, 1991). Entonces la dinámica no lineal del motor DC paralelo (5) es de **fase mínima** (Sastry y Bodson, 1989).

3.2.4 Seguimiento Acotado.

La interpretación de (18) como la dinámica que describe el comportamiento interno del sistema cuando la salida es forzada a seguir a la salida $y(t) = 0$, puede extenderse al caso en que la salida será seguida es cualquier función arbitraria. (Isidori, 1989)

El problema de seguimiento de la trayectoria de salida $y_m(t)$ es el problema de encontrar una ley de control para la entrada u tal que, no importando cual es el estado inicial del sistema en la región $\Omega \subseteq R^n$, $y(t)$ converge asintoticamente a la función de referencia prescrita $y_m(t)$ mientras que todo el estado x permanece acotada, e $y_m(t)$ y sus derivadas son también acotadas.

La proposición 2.1 en (Sastry e Isidori, 1989), (ver demostración, solo necesita mostrar que z_2 es acotada) establece que si la dinámica de cero del sistema no lineal (13) como es definido en (18) es exponencialmente estable y si $h(z_1, z_2)$ es Lipscitz en z_2, entonces aplicando la ley de control

$$v = \dot{y}_m + g\,(y_m - y), \tag{19}$$

resulta en seguimiento acotado, [esto es, $x \in R^2$ es acotado e $y(t) \to y_m(t)$], siempre que $y_m(t)$ y $\dot{y}_m(t)$ sean acotadas.

La dinámica de cero del sistema no lineal (13) es definida en (18) es exponencialmente estable, como es probada en la sección anterior.

La ecuación que representa la dinámica de $h(z_1, z_2)$ se da en (15)

$$h(z_1,z_2)=c_1c_5z_1+(c_2c_5z_1-c_3c_4)(z_2+c_5z_1)\frac{2}{c_3}+c_5c_6t_L.$$

El Jacobiano de $h(z_1, z_2)$ con el respecto a z_2 es dado por

$$\frac{\partial h}{\partial z_2} = (c_2 c_5 z_1 - c_3 c_4)\left(\frac{2}{c_3}\right),$$

Puesto que $z_1 = x_1 = w$ es acotado, el Jacobiano es acotado, entonces $h(z_1, z_2)$ es Lipschitz en z_2.

Entonces, bajo el efecto de una entrada de la forma (16) con v dado por (19) la salida del sistema sigue a la salida deseada y_R, con un error que puede hacerse converger a cero en $t \to \infty$, con un decaimiento arbitrario exponencial rápido. (Sastry e Isidory, 1989).

Capítulo 4

Control Adaptivo de Sistemas SISO linealizables.

"En el lenguaje cotidiano, adaptar significa cambiar un comportamiento para conformar una nueva circunstancia. Intuitivamente, un controlador adaptivo es un controlador que puede modificar su comportamiento en respuesta a los cambios en la dinámica del proceso y las características de la perturbación" (Aström y Wittenmark, 1995).

"El controlador adaptivo es un controlador con parámetros ajustables y un mecanismo para ajustar los parámetros" (Aström y Wittenmark, 1995).

Aunque que la teoría de los motores DC esta bien establecida (Krause, 1985; Leonhard, 1985; Fitzgerald et al., 1983) los parámetros de la descripción matemática en muchos casos son sólo aproximadamente conocidos y varían durante la operación. La técnica de control de linealización por realimentación para el motor DC paralelo requiere la cancelación de los términos no-lineales. Cuando esos términos contienen parámetros desconocidos, el desarrollo de la versión adaptiva es muy necesario.

En este capítulo, el diseño de la ley de control, usando esquemas de control adaptivo desarrollado en (Sastry e Isidory, 1989) se revisan, y se presenta la aplicación al controlador linealizado para el motor DC paralelo desarrollado en el Capítulo 3 está presente. El desempeño del algoritmo del controlador adaptivo se estudiará experimentalmente en un banco de pruebas basado en una microcomputadora desarrollada en el Laboratorio de Instrumentación de Proceso.

4.1 Linealización por Realimentación Adaptiva: Caso - Grado Relativo Uno.

En esta sección, se presenta el procedimiento de diseño de control adaptivo para un sistema linealizable dado en (Sastry e Isidory, 1989). Consideremos el sistema

$$\begin{aligned} \dot{x} &= f(x)+g(x)u \\ y &= h(x), \end{aligned} \tag{20}$$

con $L_g h(x) \neq 0$ (grado relativo uno). Asumiendo estos sistemas SISO linealizables por realimentación son lineales y de parámetro desconocido, esto es $f(x)$ y $g(x)$ tiene la forma

$$f(x)=\sum_{i=1}^{n_1} a_i f_i(x) \tag{21}$$

$$g(x)=\sum_{j=1}^{n_2} b_j g_j(x), \tag{22}$$

con $a_i, i = 1,2,\ldots n_1$, b_j, $j = 1,2,\ldots,n_2$ son parámetros desconocidos y las funciones $f_i(x)$, $g_j(x)$ son funciones conocidas. En el tiempo t, los estimados de las funciones f y g son, respectivamente

$$\hat{f}(x)=\sum_{i=1}^{n_1} \hat{a}_i(t) f_i(x) \tag{23}$$

$$\hat{g}(x)=\sum_{j=1}^{n_2} \hat{b}_j(t) g_j(x), \tag{24}$$

donde $\hat{a}_i(t)$, $\hat{b}_j(t)$ son la estimación de a_i, b_j respectivamente en el tiempo. Consiguientemente la ley de control adaptiva u, para la linealización adaptiva es reemplazada por

$$u=\frac{1}{\hat{L}_g h}\left(-\hat{L}_f h+v\right) \tag{25}$$

donde $\hat{L_g}h, \hat{L_f}h$ son los estimados de $L_g h$, L_f h respectivamente. Usando (23) y (24) resulta en

$$\hat{L_f}h = \sum_{i=1}^{n_1} \hat{a}_i(t) L_{f_i} h \tag{26}$$

$$\hat{L_g}h = \sum_{j=1}^{n_2} \hat{b}_j(t) L_{g_j} h \tag{27}$$

Definimos $q = [a^T, b^T] \in R^{n_1+n_2}$ para ser el parámetro vector **verdadero** y $\hat{q} = [\hat{a}^T, \hat{b}^T] \in R^{n_1+n_2}$ la estimación del parámetro, y $e_q = q - \hat{q}$ el error del parámetro. Puesto que el sistema tiene grado relativo uno, usando (20) resulta en

$$\dot{y} = L_f h + (L_g h) u \tag{28}$$

Reemplazando (25) en (28)

$$\dot{y} = v + e_a^T w_1 + e_b^T w_2, \tag{29}$$

donde

$$w_1 \in R^{n_1} = \begin{bmatrix} L_{f1} h \\ \vdots \\ L_{fn1} h \end{bmatrix} \tag{30}$$

$$w_2 \in R^{n_2} = \begin{bmatrix} L_{g1}h \\ \vdots \\ L_{g\,n2}h \end{bmatrix} \frac{\left(-\hat{L_f}h + v\right)}{\hat{L_g}h}, \qquad (31)$$

la ley de control adaptivo usado para seguimiento es

$$v = \dot{y}_m + g\,\left(y_m - y\right)$$

Esta resulta en la siguiente ecuación de error que relaciona $y - y_m = e_y$ al error del parámetro $e_q = \left(e_a^T, e_b^T\right)$

$$\dot{e}_y + g\,e_y = e_q^T\,w,$$

donde $w \in R^{n_1+n_2}$ es la concatenación de w_1 , w_2 , y_m es la salida del modelo de referencia, y $g > 0$.

Considere que la descripción en estado-espacio para el sistema de modelo de referencia SISO es especificado por:

$$\dot{x}_m = A_m x_m + B_m r$$
$$y_m = C_m^T x_m,$$

donde y_m *(t)* es la salida del modelo de referencia lineal invariante en el tiempo y *r (t)* es la entrada. El grado relativo del modelo de la referencia debería ser mayor ó igual al grado relativo *r* del sistema no-lineal.

Teorema 1 *(seguimiento Adaptivo): Considere un sistema no-lineal de fase mínima de la forma (20), con las suposiciones sobre f, g dadas en (21)-(24). Definimos la ley de control*

$$u=\frac{1}{\hat{L_g}h}\left(-\hat{L_f}h+\dot{y}_m+g\,(y_m-y)\right) \qquad (32)$$

Asuma que $\hat{L_g}h$ *como definido por (27) es acotada lejos de cero.*

Entonces, si $y_m(t)$ *es acotada, la ley de actualización del parámetro*

$$\dot{e_q}=-(y-y_m)w=-e_y\,w, \qquad (33)$$

Resulta acotada, y(t) converge asintoticamente a $y_m(t)$*. Adicionalmente, todos las variables de estado x (t) de (20) son acotadas.*

Para una Demostración: ver (Sastry y Boston, 1989)

4.2 Seguimiento adaptivo de Velocidad para el Motor DC paralelo.

En esta sección, usaremos el procedimiento de diseño dado en la sección anterior para diseñar un controlador adaptivo de seguimiento de velocidad de un motor DC paralelo. Considere (5) en términos de parámetros desconocidos α_i para i = 1, 2, 3, 4 y β_j para j = 1, 2

$$\dot{x}=f(x)+g(x)u$$
$$y=h(x)=x_1$$

$$f(x)=\begin{bmatrix}-a_1x_1-a_2x_1x_2^2-a_4\\ -a_3x_2\end{bmatrix},g(x)=\begin{bmatrix}b_1x_2\\ b_2\end{bmatrix} \qquad (34)$$

donde

$$a_1=c_1,a_2=c_2,a_3=c_4,a_4=c_6t_L,b_1=c_3,b_2=c_5.$$

hacemos

$$f_1(x) = \begin{bmatrix} -x_1 \\ 0 \end{bmatrix};\ f_2(x) = \begin{bmatrix} -x_1 x_2^2 \\ 0 \end{bmatrix};\ f_3(x) = \begin{bmatrix} 0 \\ -x_2 \end{bmatrix};$$

$$f_4(x) = \begin{bmatrix} -1 \\ 0 \end{bmatrix},\ g_1(x) = \begin{bmatrix} x_2 \\ 0 \end{bmatrix};\ g_2(x) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Entonces

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x) + a_4 f_4(x)$$
$$g(x) = b_1 g_1(x) + b_2 g_2(x).$$

La estimación de $f(x)$ y $g(x)$ son dadas por

$$\hat{f}(x) = \hat{a}_1 f_1(x) + \hat{a}_2 f_2(x) + \hat{a}_3 f_3(x) + \hat{a}_4 f_4(x)$$
$$\hat{g}(x) = \hat{b}_1 g_1(x) + \hat{b}_2 g_2(x).$$

La estimación de $L_g h, L_f h$ son

$$\hat{L_f} h = \hat{a}_1 L_{f1} h + \hat{a}_2 L_{f2} h + \hat{a}_3 L_{f3} h + \hat{a}_4 L_{f4} h$$
$$\hat{L_g} h = \hat{b}_1 L_{g1} h + \hat{b}_2 L_{g2} h.$$

Las derivadas Lie de f_i, g_j, para $i = 1, 2, 3, 4,$ y $j = 1, 2$ son dadas como sigue

$$L_{f\,1}h = \frac{\partial h}{\partial x} f_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -x_1 \\ 0 \end{bmatrix} = -x_1$$

$$L_{f\,2}h = \frac{\partial h}{\partial x} f_2 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -x_1 x_2^2 \\ 0 \end{bmatrix} = -x_1 x_2^2$$

$$L_{f\,3}h = \frac{\partial h}{\partial x} f_3 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -x_2 \end{bmatrix} = 0$$

$$L_{f\,4}h = \frac{\partial h}{\partial x} f_4 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = -1$$

$$L_{g\,1}h = \frac{\partial h}{\partial x} g_1 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ 0 \end{bmatrix} = x_2$$

$$L_{g\,2}h = \frac{\partial h}{\partial x} g_2 = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0,$$

entonces

$$\hat{L_f} h = -\hat{\alpha}_1 x_1 - \hat{\alpha}_2 x_1 x_2^2 - \hat{\alpha}_4 \qquad (35)$$

$$\hat{L_g} h = \hat{b}_1 x_2 \qquad (36)$$

y

$$\dot{y} = v + e_a^T w_1 + e_b^T w_2 = v + e_q^T w,$$

donde

$$w_1 \in R^4 = \begin{bmatrix} L_{f1}h \\ L_{f2}h \\ L_{f3}h \\ L_{f4}h \end{bmatrix} = \begin{bmatrix} -x_1 \\ -x_1 x_2^2 \\ 0 \\ -1 \end{bmatrix}$$

$$w_2 \in R^2 = \begin{bmatrix} L_{g1}h \\ L_{g2}h \end{bmatrix} \frac{(-\hat{L_f} h + v)}{\hat{L_g} h} = \begin{bmatrix} \frac{-(-\hat{\alpha}_1 x_1 - \hat{\alpha}_2 x_1 x_2^2 - \hat{\alpha}_4) + v}{\hat{b}_1} \\ 0 \end{bmatrix}$$

Puesto que la dinámica de ceros del sistema no-lineal es exponencialmente estable y $n(z_1, z_2)$ es Lipschitz en z_1 (ver Capítulo 3) es posible usar la siguiente ley de control para seguimiento.

$$v = \dot{y}_m + g\,(y_m - y),$$

Lo cual da como resultado la siguiente ecuación de error de seguimiento de salida

$$\dot{e}_y + g\ e_y = e_q^T\, w, \tag{37}$$

donde $w \in R^{n_1+n_2}$ se define por la concatenación de w_1, w_2, $e_y = y - y_m$ y

Considere $x_{m\,1} = y_m$ y $x_{m\,2} = \dot{y}_m$, el modelo de la referencia usado para el sistema adaptivo de control es un modelo de segundo orden, entonces su representación en variables de espacio-estado esta dada por

$$\begin{aligned} \dot{x}_m &= \begin{bmatrix} 0 & 1 \\ -b & -a \end{bmatrix} x_m + \begin{bmatrix} 0 \\ k_p \end{bmatrix} r \\ y_m &= \begin{bmatrix} 1 & 0 \end{bmatrix} x_m . \end{aligned} \tag{38}$$

Donde los valores de parámetros $-b, -a$ y k_p se eligen para satisfacer las siguientes especificaciones de desempeño:

Tiempo de establecimiento (T_s) = 10seg, porcentaje de sobreimpulso (P.O.) < 5%.

Para el modelo de referencia subamortiguado $(z = 0.707)\ a = 0.8, b = 0.32$ y k_p = 0.48, y para el modelo de referencia sobreamortiguado (ζ = 1) a = 0.8, b = 0.16 y k_p = 0.24 han sido escogidas.

Por los resultados de Capítulo 3 la dinámica de ceros es globalmente exponencialmente estable (sistema de fase mínima) y

$$\hat{L_g} h = \hat{b}_1 x_2,$$

es acotado puesto que $x_2 = i_F$ es acotado, también y_m, $x_1 = w$ son acotados.

Entonces usando la ley de actualización de parámetros estimados

$$e_q = -\hat{q} = -e_y \begin{bmatrix} -x_1 \\ -x_1 x_2^2 \\ 0 \\ -1 \\ \dfrac{x_2(-(-\hat{a}_1 x_1 - \hat{a}_2 x_1 x_2^2 - \hat{a}_4) + v)}{\hat{b}_1 x_2} \\ 0 \end{bmatrix}$$

con la ley de control siguiente

$$u = \frac{1}{\hat{b}_1 x_2}(-(-\hat{a}_1 x_1 - \hat{a}_2 x_1 x_2^2 - \hat{a}_4) + \dot{y}_m + g(y_m - y)), \qquad (39)$$

resulta que $y(t)$ es acotada converge asintoticamente a $y_m(t)$.

El diagrama de bloques del controlador linealizante adaptivo se muestra en la Figura 5.

El controlador consiste en cuatro de partes: una **planta** que contiene parámetros desconocidos, un **modelo de referencia** que define la salida deseada del sistema de control, una **ley de control** de realimentación que contiene los parámetros ajustables, y **la ley** de **actualización** que ajusta parámetros del controlador a fin de lograr error cero entre la salida del modelo y la salida de la planta.

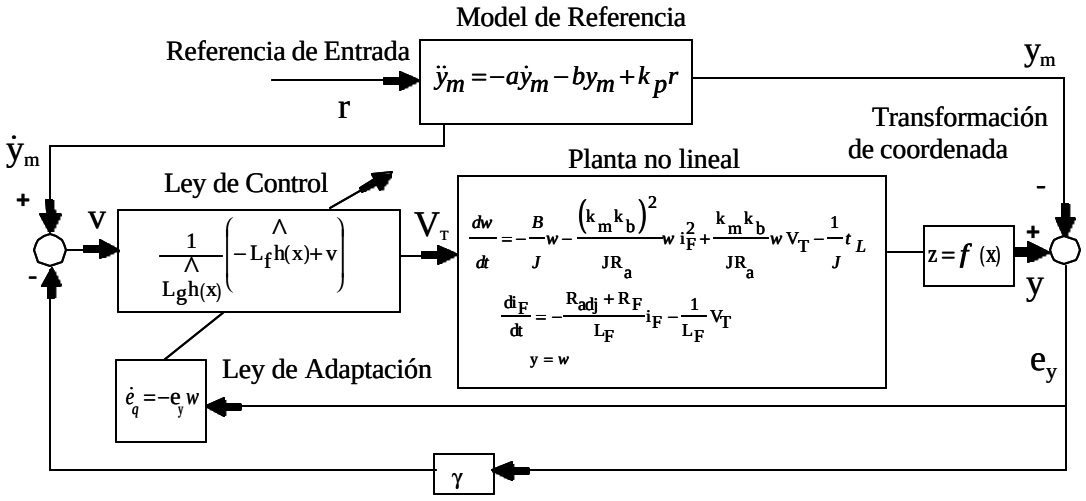


Figura 5: Diagrama de Bloques del Controlador Linealizante Adaptivo

4.3 Observador no-lineal de Torque de Carga.

En la sección previa el torque de carga se presume constante, pero en el sistema actual los cambios de carga necesitan ser considerados.

Usamos la teoría desarrollada en (Krener e Isidori, 1983) para construir un observador no-lineal con dinámica de error lineal para estimar el torque de carga t_L midiendo la corriente de campo i_F, y la velocidad w. Consideramos que el torque de carga t_L, cambia muy lentamente durante el período de muestreo ($\frac{dt_L}{dt} = 0$).

Considere el sistema descrito por (34) con el estado aumentado $x_3 = \alpha_4$ y salida $y_2 = x_2 = i_F$

$$x_{aug} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}.$$

Asuma

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

serán la salidas medibles, entonces el modelo aumentado del sistema es dado por

$$\begin{aligned} \dot{x}_1 &= -a_1 y_1 - a_2 y_1 y_2^2 + b_1 y_2 u - x_3 \\ \dot{x}_2 &= -a_3 y_2 + b_2 u \\ \dot{x}_3 &= 0 \\ y_1 &= x_1 \\ y_2 &= x_2. \end{aligned}$$

Escribiendo la expresión anterior en una forma más compacta

$$\begin{aligned} \dot{x}_{aug} &= A x_{aug} + j\,(y,u) \\ y &= C x_{aug}, \end{aligned} \tag{40}$$

donde

$$A = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}; \; C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}; \; j\,(y,u) = \begin{bmatrix} -a_1 y_1 - a_2 y_1 y_2^2 + b_1 y_2 u \\ -a_3 y_2 + b_2 u \\ 0 \end{bmatrix}$$

El observador no lineal es dado por la expresión siguiente

$$\begin{aligned} \dot{\hat{x}}_{aug} &= A\hat{x}_{aug} + j\,(y,u) + L\,(y - \hat{y}) \\ \hat{y} &= C\hat{x}_{aug}. \end{aligned} \tag{41}$$

Definiendo el error

$$e = x_{aug} - \hat{x}_{aug}, \tag{42}$$

como la diferencia entre el estado aumentado y su estimado. Reemplazando (40) y (41) en (42) se obtiene la dinámica de error lineal

$$\dot{e} = (A - LC)\, e.$$

Puesto que

$$rank = \begin{bmatrix} C \\ CA \end{bmatrix} = 3,$$

el par *(C, A)* es observable, entonces podemos asignar los autovalores de *A-LC* arbitrariamente, de modo que $e(t) \to 0$ y $\hat{x}_3 \to a_4$ (Kailath, 1980) arbitrariamente rápido.

Suponiendo un tiempo de establecimiento de 0.4 seg y un factor de amortiguamiento de 0.707 para la dinámica del observador, los polos de lazo cerrado del observador son

$$\begin{aligned} s_1 &= -10 + j10 \\ s_2 &= -10 - j10 \\ s_3 &= -1. \end{aligned}$$

La ganancia L seleccionada, la cual permite colocar las raíces de $|sI - (A - LC)| = 0$ en los valores antes mencionado es dado por

Por lo tanto la dinámica del observador será

$$\begin{aligned} \dot{\hat{x}}_1 &= -\hat{x}_3 - a_1 y_1 - a_2 y_1 y_2^2 + b_2 y_2 u + 10(y_1 - \hat{x}_1) - 10(y_2 - \hat{x}_2) \\ \dot{\hat{x}}_2 &= -a_3 y_2 + b_2 u + 10(y_1 - \hat{x}_1) + 11(y_2 - \hat{x}_2) \\ \dot{\hat{x}}_3 &= -10(y_1 - \hat{x}_1) + 9(y_2 - \hat{x}_2) \\ y &= \hat{x}_3 . \end{aligned} \tag{43}$$

Capítulo 5

Estudio de Simulación para el Algoritmo de Control.

El enfoque propuesto ha sido probado por simulación usando el SIMULINK™, este programa es una extensión a MATLAB™. Los valores de los parámetros usados para la simulación son aquellos mostrados en la Tabla 3.

El valor nominal de la corriente de armadura es de 1A y el valor nominal de velocidad es 1,725 *RPM* ó 180.6 *rad/seg.* El valor nominal de la corriente de armadura resulta en un torque nominal de 0.69 *N-m*. En la implementación real, no es posible lograr este valor de velocidad debido a la saturación del amplificador PWM a 100 V, por lo tanto usaremos una velocidad menor de 1,432.4 *RPM* (150 *rad/seg*) como la máxima velocidad alcanzable del rotor, en ese caso el torque de carga es menor que el valor de nominal, y elegimos el torque de carga máximo $t_L = 0.1\ N\text{-}m$ en las simulaciones.

Tabla 3: Parámetros del Motor DC Bobinado

Parámetros	**Valores**
Potencia	125 Watts
Voltaje	125 Voltios
Velocidad de Rotor	1725 RPM
Corriente de Armadura	1.0 Amperios
Corriente de Campo	0.44 Amperios
Resistencia de Armadura Inductancia de Armadura	15.9 Ω 18.7 mH
Resistencia de Campo	267 Ω
Inductancia de Campo	24.736 H
Coeficiente de Fricción	0.0023 N-m/rad/seg
Constante Back-EMF	0.69 Volts/rad/seg

5.1 Capacidad Adaptiva de Seguimiento.

El diagrama de bloques en SIMULINK™ para el sistema es dado en la Figura 6. El modelo emplea un bloque de escalón de entrada para simular el comando de velocidad de referencia y el bloque de secuencia para simular el torque de carga t_L aplicada al eje del rotor. Las salidas del sistema son la velocidad del rotor y la corriente de campo. El torque de carga de las perturbaciones t_L se variaron como

$$\begin{aligned} t_L &= 0 && 0 \le t \le 15 \\ t_L &= 0.1(t-15)/5 && 15 \le t \le 20 \\ t_L &= 0.1 && 20 \le t \end{aligned}$$

El límite sobre la velocidad es $w_{0\,max}$ = *150 rad/seg*. El voltaje de entrada u en el motor se restringe a $0 \le u \le 100$ Voltios.

Los bloques del motor, mo_ref, tL_obs y la adap_law son bloques en el formato de SIMULINK que define la dinámica del motor DC paralelo, modelo de la referencia, estimación del torque de carga y el controlador linealizante adaptivo.

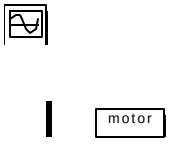


Figure 6: Diagrama de Bloques para la Simulación del Controlador Linearizante Adaptivo.

Estos bloques tienen un código S-función correspondiente con el mismo nombre como el nombre de modelo. Esta función-S interactúa con el SIMULINK para la simulación y análisis. Una función-S es una función que define la dinámica del modelo, estas son funciones de MATLAB con una sintaxis especial de llamada, la cual nos permite modelar ecuaciones dinámicas. Estos bloques se han implementado como funciones-S en código C y han sido compilados como archivos MEX usando el compilador Watcom C/C++386 versión 10.0. Vea el manual del SIMULINK para más información sobre funciones-S.

Comentario 3: *Mientras que la función-S puede implementarse como un archivo-M ó archivo-MEX para la simulación, se uso el archivo-MEX, puesto que es compatible con la estructura de datos de la interfase API (programa de aplicación en SIMULINK Real-Time Workshop), llamado un SimStruct, que encapsulan todos los datos para cada instancia del modelo. Usando el formato función-S definido por SIMULINK 1.3 con las mismas estructuras que API hace posible generar códigos que automáticamente incorporan cualquier función-S que están en el modelo SIMULINK. Esto simplifica el proceso de incorporación de código C escrito dentro del programa en tiempo real.*

motor: El modelo motor que representa la dinámica no-lineal de motor DC paralelo ha sido escrito como un archivo-C MEX.

mo_ref: El modelo de referencia que representa la dinámica para el seguimiento se ha escrito como un archivo-C MEX.

tL_obs: El modelo de observador de torque de carga que representa la dinámica del observador no-lineal con la dinámica de error lineal ha sido escrito como un archivo-C MEX.

adap_law: Este modelo implementa el controlador linealizante adaptivo, este modelo ha sido escrito como un archivo-C MEX.

El procedimiento para construir un bloque SIMULINK es el siguiente:

- Escriba un código C que representa la dinámica del sistema usando formato archivo-MEX función-S, incluya el archivo C simulink.h en su código C, esto proporciona la interfase del archivo-MEX estándar a Matlab.

- El archivo C creado puede convertirse a un archivo ejecutable usando la interfase de archivo-MEX estándar. Por ejemplo, sobre una máquina en DOS, el comando es
cmex nombre del archivo.c

Desde el prompt del Matlab el comando es
!cmex nombre del archivo.c

- Transformar el archivo-MEX generado en un bloque usando un bloque de comandos de funciones-S desde el menú de opciones del SIMULINK.

Los archivos-C MEX motor.c, mo_ref.c, tL_obs.c y adap_law.c se han convertido a bloques de funciones-S que usan el procedimiento antes mencionado. El código C para estas funciones-S se muestran en el Apéndice A.

Los parámetros de simulación se muestran en la Tabla 4. El desempeño del sistema se evalúa bajo dos condiciones diferentes sobre la dinámica del modelo de referencia estudiado.

Para los dos casos se ha usado la misma variación de entrada de torque de carga como se muestra en la Figura 7.

Existe un límite de 100 V sobre el voltaje de salida máximo disponible en el amplificador usado para la implementación en tiempo real. Por lo tanto, se limitara a 100 V al voltaje de entrada sobre el motor en simulaciones del sistema adaptivo.

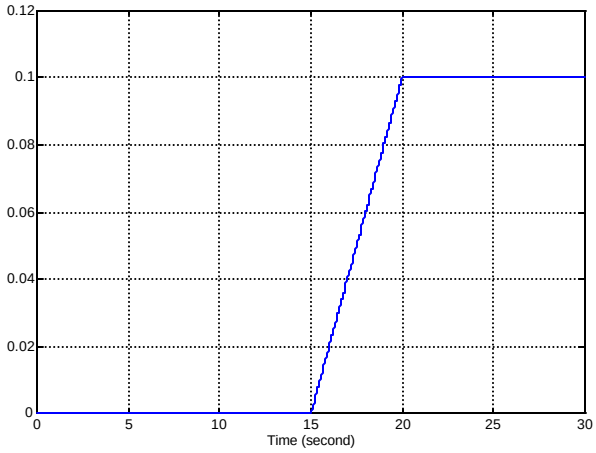


Figure 7: Variación del torque de carga de entrada.

5.1.1 Caso 1. Respuesta Críticamente Amortiguada.

Para este caso, se impuso el limite de 100 V sobre el voltaje de salida y se uso la respuesta críticamente amortiguada ($\zeta = 1$) para el modelo de referencia que representa la trayectoria de velocidad para el seguimiento. La Figura 8 presenta la respuesta de velocidad al comando de escalón hacia arriba de velocidad, donde la línea entrecortada indica la trayectoria de la referencia, la línea sólida corresponde a la velocidad del rotor del motor DC paralelo y la línea punteada representa el comando de velocidad. Podemos ver un perfecto seguimiento del modelo de referencia, la trayectoria de velocidad alcanza la velocidad deseada de 150 *rad/seg* en el tiempo de $t = 20$ seg aproximadamente.

Tabla 4: Parámetros de Simulación

Parametros	**Valores**
Tiempo de Simulación	30 *seg*
Método de Integración	Runge-Kutta de quinto orden
Dimensión de paso mínimo	0.001
Dimensión de paso Máximo	10
Tolerancia	$1e^{-4}$

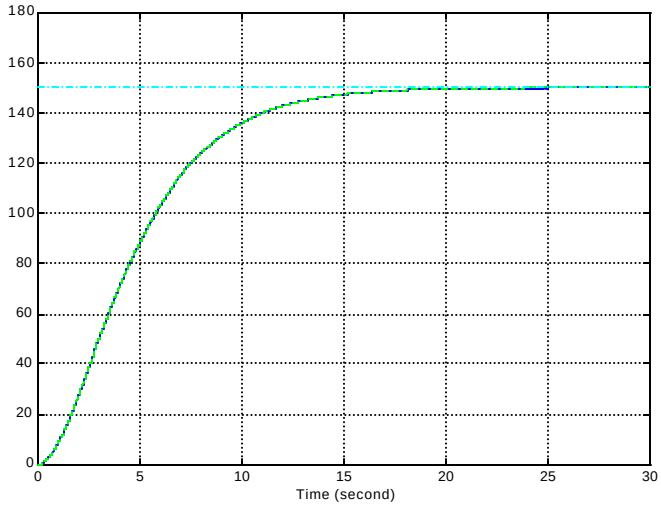


Figure 8: Velocidad de Respuesta en rad/seg Vs. Tiempo en segundos (ζ = 1.0)

Notamos que la velocidad del motor sigue el modelo de referencia cuando el torque de carga se varía linealmente desde 0 a 0.1 N - m en el intervalo entre 15 a 20 *segundos.*

La Figura 9 muestra el barrido de voltaje de salida del controlador (entrada en el motor), el voltaje de salida alcanza el nivel de saturación de 100 Voltios en t = 5 seg y permanece a este valor hasta aproximadamente en t = 8 seg, pero el efecto sobre la respuesta de velocidad en insignificante como se muestra en la Figura 8. La Figura 10 se muestra el torque de carga estimado $\hat{\boldsymbol{t}}_L$, nota que el observador presenta una buena respuesta de estado estable.

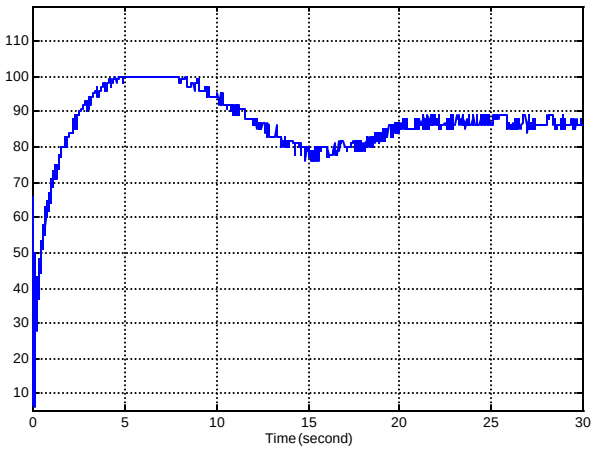


Figura 9: V_T en voltios Vs. Tiempo en segundos

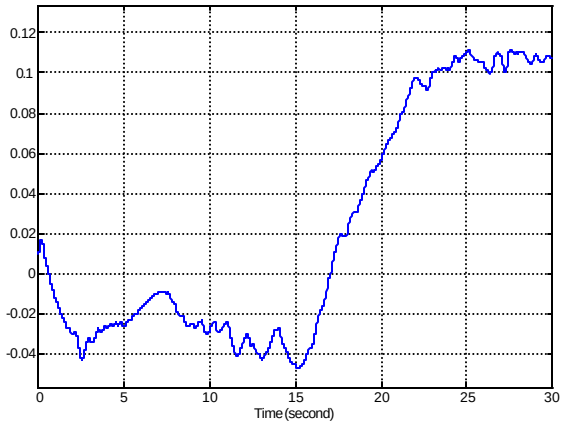


Figure 10: Estimación de torque de carga para $\zeta = 1.0$

5.1.2 Caso 2. Respuesta Sub-amortiguada.

Ahora imponemos el mismo límite de 100 Voltios sobre el voltaje de salida, pero usamos respuesta subamortiguada ($\zeta = 0.707$) para el modelo de la referencia. La respuesta de velocidad al comando de velocidad se muestra en que la Figura 11, donde la línea entrecortada indica la trayectoria de la referencia, la línea sólida corresponde a la velocidad del rotor del motor DC paralelo y la línea punteada representa el comando de velocidad.

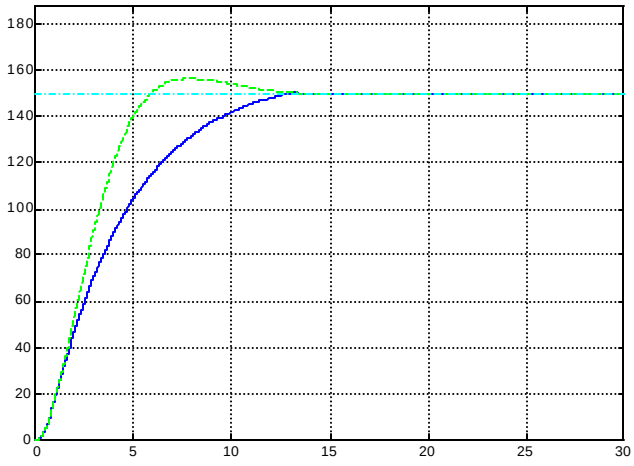


Figura 11: Respuesta de Velocidad para $\zeta = 0,707$

De esta respuesta, se puede ver que la velocidad del motor no sigue la respuesta del modelo de la referencia. La degradación de la respuesta es debido a la saturación del amplificador D.C. a 100 V. Aunque la respuesta transitoria no corresponde inicialmente a la respuesta de modelo de la referencia, esta alcanza la velocidad deseada de 150 *rad/seg* en un tiempo aproximadamente t = 17 seg. Note también que la velocidad del motor sigue el modelo de la referencia cuando el torque de carga se varía de modo lineal desde 0 a 0.1 *N-m* en el intervalo entre 15 a 20 *segundos.*

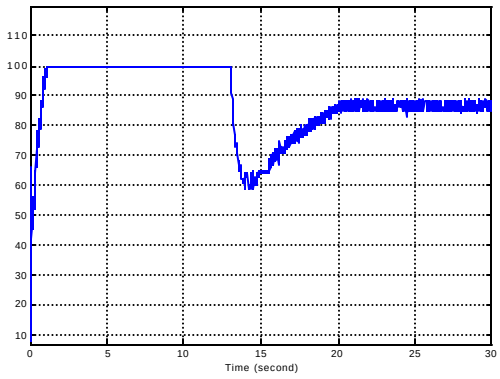


Figure 12: Barrido de Voltaje del controlador para $\zeta = 0.707$

La Figura 12 muestra el barrido de voltaje de salida del controlador (entrada del motor), el voltaje de salida alcanza el nivel de saturación de 100 voltios en t = 1 seg y permanece en este valor hasta aproximadamente t = 13 seg, esta saturación tiene efecto solamente en la respuesta transitoria de la velocidad como se muestra en la Figura 11. La figura 13 muestra el torque de carga estimado $\hat{t}_L$.

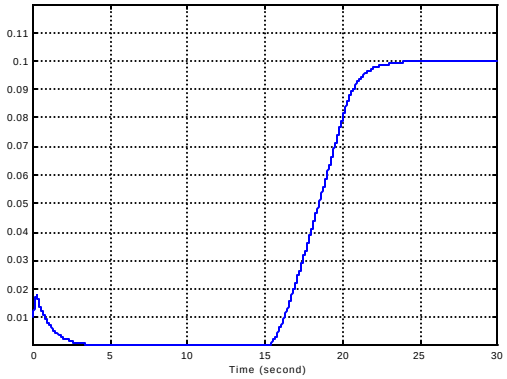


Figure 13: Estimación de torque de carga para $\zeta = 0.707$

El observador presenta una mejor respuesta que los casos anteriores, esto se debe a que la salida permanece constante por un tiempo muy largo.

De los resultados de la simulación, nosotros podemos ver que el controlador linealizante adaptivo basado en la técnica de geometría diferencial puede usarse efectivamente para el diseño de un controlador no-lineal para un motor DC paralelo. Las pruebas de simulación muestran un comportamiento muy satisfactorio del controlador propuesto y su robustez bajo la presencia de incertidumbre de parámetros de la máquina.

En el algoritmo del controlador, existe un **punto de singularidad** cuando la corriente de campo i_F = 0 como se describió en el Capítulo 4, para prevenir este caso, se ajusta x_2 en el algoritmo del controlador y en el observador a $x_2 = \max(0.001, |i_F|) \times sign(i_F)$.

Las Figura 14 muestra el error en la estimación del parámetro para el Caso 1, y las Figura 15 muestra el error en la estimación de parámetro para el Caso 2, donde la línea sólida corresponde a los errores de parámetro e_{q1} y e_{q4}, la línea entrecortada corresponde al error de parámetro e_{q2} y e_{q5}, y la línea punteada representa al error de parámetro e_{q3} y e_{q6}.

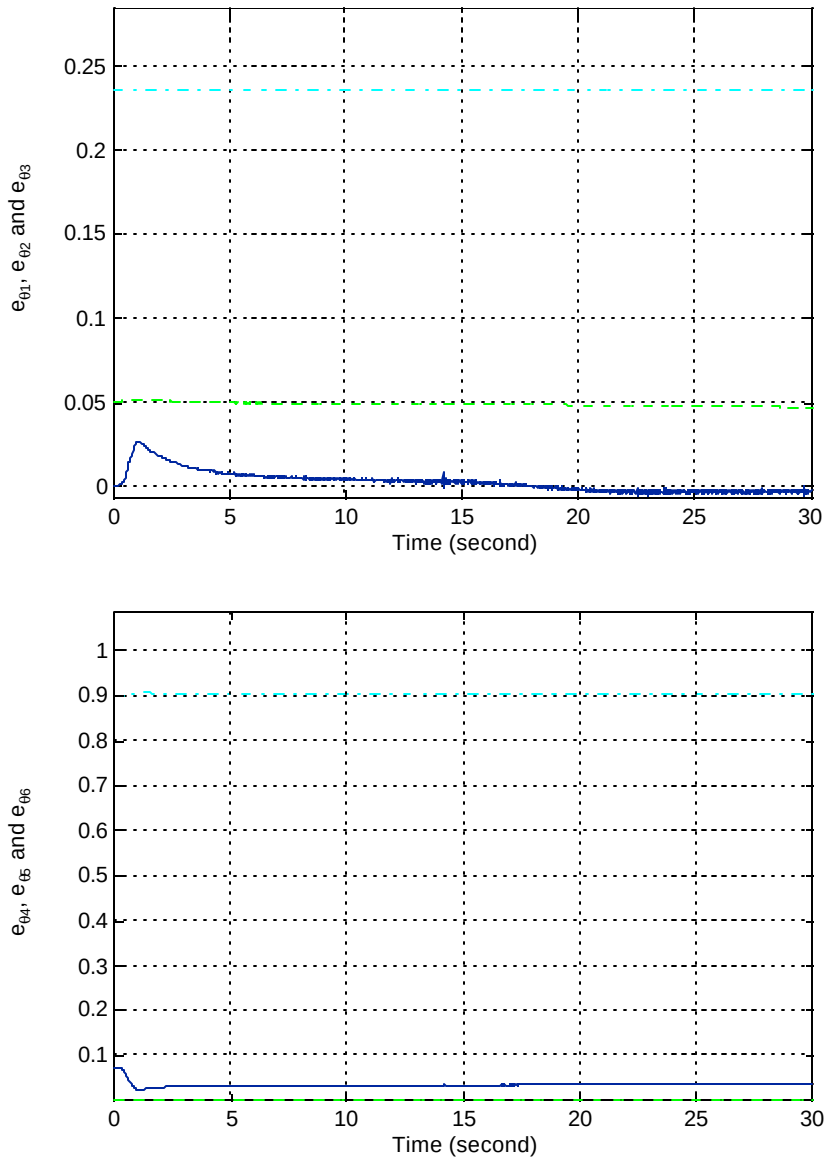


Figura 14: Error de Estimación de Parámetros, $e_{\theta 1}$, $e_{\theta 2}$, $e_{\theta 3}$, $e_{\theta 4}$, $e_{\theta 5}$ y $e_{\theta 6}$ (ζ = 1. 0)

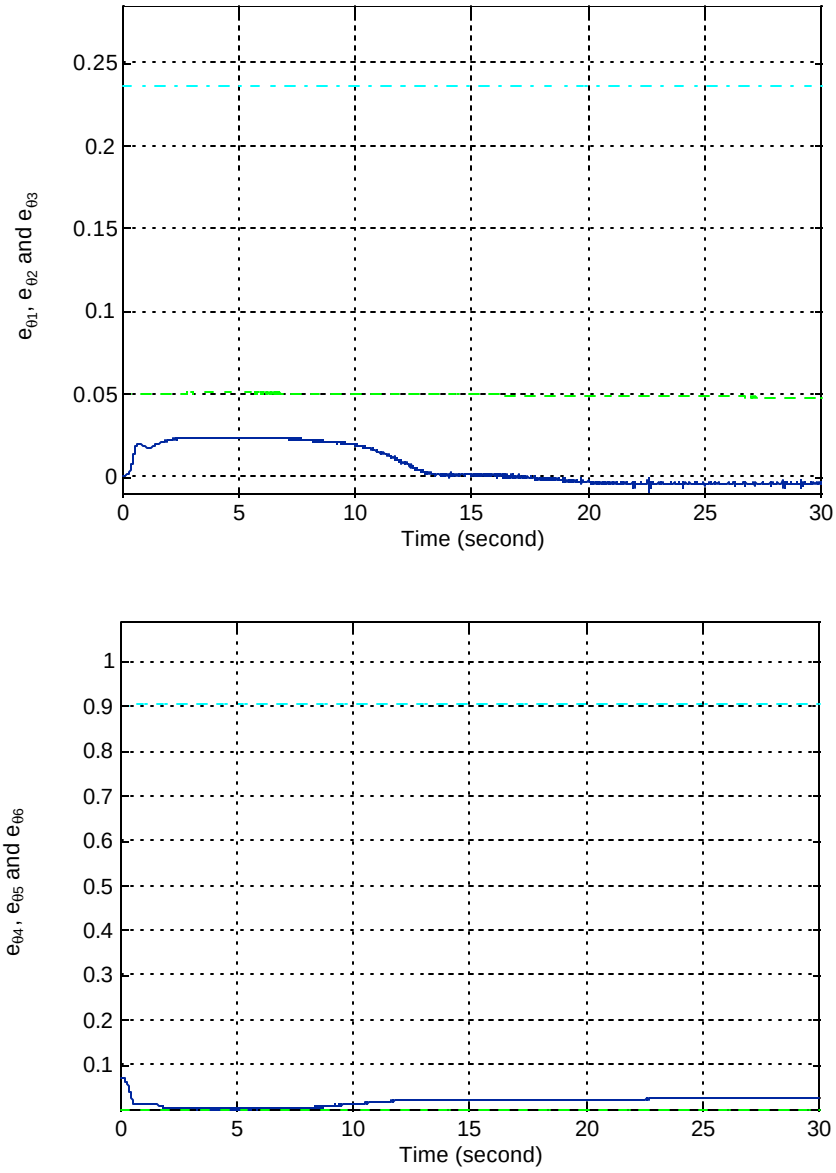


Figura 15: Error de Estimación de Parámetros, $e_{\theta1}$, $e_{\theta2}$, $e_{\theta3}$, $e_{\theta4}$, $e_{\theta5}$ y $e_{\theta6}$ ($\zeta = 0.707$)

Los valores verdaderos de los parámetros son:

$$a_1 = 0.011,\ a_2 = 3.9308,\ a_3 = 16.7367,\ a_4 = 1,\ b_1 = 1.5723,\ b_2 = 0.04043$$

Los valores iniciales de $\hat{a}$ y $\hat{b}$ se escogen como

$$\hat{a}_{01} = 0.01,\ \hat{a}_{02} = 3.9,\ \hat{a}_{03} = 16.5,\ \hat{a}_{04} = 0.1,\ \hat{b}_{01} = 1.5,\ \hat{b}_{02} = 0.04$$

Indicando conocimiento de parámetros a priori. Podemos ver mientras el error de seguimiento de velocidad converge a cero para ambos casos, el error de parámetro no. La razón para la no convergencia del error de parámetro puede explicarse como se sigue; puesto que la señal de referencia *r(t)* es constante, es posible que muchos vectores de los parámetros del controlador, a parte del parámetro del vector ideal, permitan la convergencia de error de seguimiento. Entonces la ley de adaptación de parámetro no necesitará averiguar los parámetros ideales.

Los parámetros estimados no convergerán a los parámetros ideales del controlador a menos que que la señal de referencia *r(t)* satisfaga la condición de *excitación persistente*. Ver (Slotine y Li, 1989) y (Sastry y Bodson, 1989) para una definición de condición de excitación persistente.

Capítulo 6

Control de Velocidad en Tiempo-Real del Motor DC paralelo.

En este Capítulo, se muestra como el controlador linealizante adaptivo desarrollado en el Capítulo 4 y probado mediante simulación en el Capítulo 5 se construyó y fue evaluado experimentalmente en el Laboratorio de Control y Automatización de la PUCP. El sistema experimental consiste de una PC equipada con una tarjeta de entrada-salida analógica y digital, un sistema de acondicionamiento de señal, y un equipo amplificador de potencia. El sistema experimental y las herramientas usadas para desarrollar el controlador en tiempo real permitió probar con éxito este trabajo de tesis, las herramientas generadas pueden ser usada por otros usuarios para realizar investigaciones en las áreas de la teoría de control moderno y control adaptivo.

El control en tiempo real se ha diseñado usando el SIMULINK Real-Time Workshop™. Este es un ambiente de generación automático de código en lenguaje C para SIMULINK. Este produce código C desde el SIMULINK y automáticamente construye un programa que puede ejecutarse en tiempo real.

6.1 Montaje Experimental.

El diagrama de bloque del sistema experimental se muestra en la Figura 16. Este sistema consiste de computadora personal con una tarjeta de adquisición de datos Lab PC + de National Instruments, módulos de acondicionamiento de señal, un amplificador de potencia PWM con la fuente de alimentación para comandar el motor, un motor DC paralelo, un tacómetro y un generador DC de imán permanente.

Las características de sistema son como se indica a continuación:

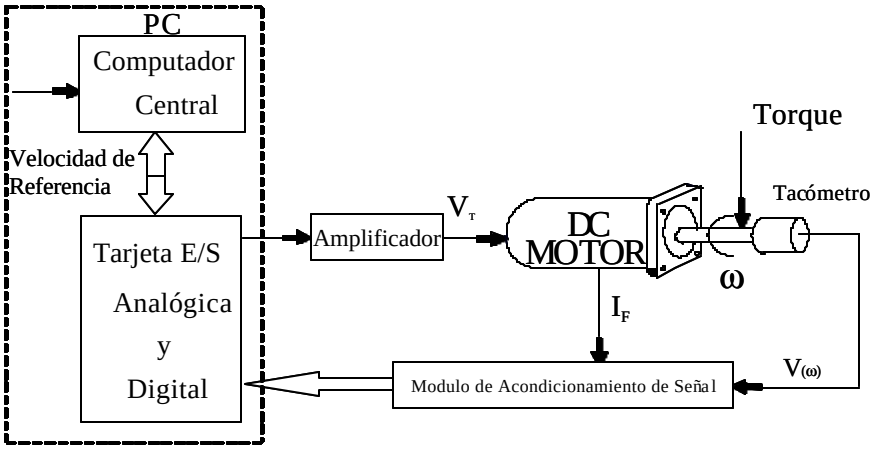
- **Planta de Control (Motor DC).** La planta de control es un motor DC de bobinado de la Compañía Reliance Electric con los valores nominales siguientes; 1/8 KW, 125 Voltios DC; 1725 RPM, Corriente de armadura de 1.0 Amperio, Corriente de campo de 0.44 Amperios.

- **Computadora Central.** Gateway 2000, 486DX2-66 En este sistema experimental, la computadora se usa para leer el voltaje, corriente, y realimentación de velocidad y basado en estas medidas evaluar el algoritmo de control.

- **Tarjeta de E/S Analógica y Digital.** Esta interfase consiste de canales de entrada-salida analógica y digital Los canales de entrada analógica se usan para muestrear valores discretos de la corriente de campo y el voltaje de salida del tacómetro, y almacenar estos datos en la memoria de la computadora. El canal de salida analógico acepta una secuencia de dígitos desde la computadora y envía un voltaje de salida analógico al amplificador DC, el cual controla el voltaje de entrada del motor.

- **Módulo de Acondicionamiento de Señal.** El módulo consiste de sensado de señales y un subsistema de acondicionamiento de señal para la adquisición y subsiguiente procesamiento de la corriente, el voltaje y las señales de velocidad.

- **Driver.** Esta unidad consiste en amplificador de potencia Copley 421 PWM DC para la transferencia de potencia al motor.



Controlador Central

Figure 16: Diagrama de Bloques del Montaje Experimental.

6.1.1 Adquisición de datos.

La tarjeta de adquisición de datos de National Instruments LabPC + se usa para la adquisición de datos. El LabPC+ contiene un conversor analógico/digital por aproximaciones sucesivas de 12 bits, esto puede ser configurado como ocho entradas analógicas de modo común, o cuatro canales de entrada diferencial. La tarjeta LabPC+ también tiene dos conversores digital/analógico de 12 bits con voltajes de salida, 24 líneas de entrada/salida (E/S) compatible con TTL, y seis canales contador/temporizador de 16 bits para temporización de entradas/salidas.

Configuración de la entrada.

Dos canales se usan para la entrada, el Canal 0 para el voltaje proporcional a la corriente de campo, y el Canal 1 para el voltaje que es proporcional a la velocidad del rotor. Puesto que se tiene la referencia a tierra, las fuentes de señal de modo común, el LabPC+ se ha configurado en la configuración de entrada **NRSE** como se muestra en la Figura 17. Ver manual del LabPC+ para detalles de configuración. Las señales son conectadas a las entradas positivas del amplificador de instrumentación del LabPC+ (entradas 1 y 2) y la señal local referenciada a tierra es conectada a la entrada negativa del amplificador de instrumentación del LabPC+.

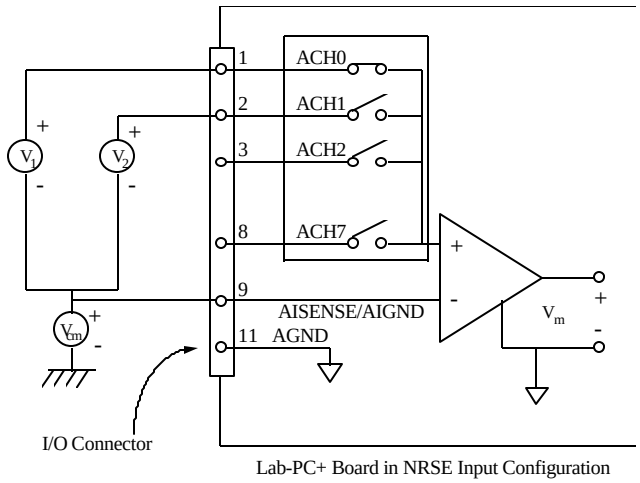


Figura 17: Conexiones de entrada de modo común para fuentes de señal con conexión a tierra.

De esta manera cualquier diferencia de potencial entre la tierra del LabPC+ y la tierra de la señal aparece como una señal de modo común en las entradas positiva y negativa del amplificador de instrumentación, y por lo tanto rechazado por el amplificador. El rango de la señal de entrada usada es unipolar desde 0 a 10 voltios.

Conversiones A/D.

La tarjeta LabPC+ se ha programado en el modo controlado de adquisición, tres conversiones A/D de dos canales de entrada se ejecutan en el intervalo de muestreo, el tiempo entre las sucesivas conversiones del A/D se ajusta a 500 μseg, los canales son barridos en la secuencia siguiente:

Canal 1, Canal 0, Canal 1, Canal 0, Canal 1, Canal 0. El formato binario usado es binario directo.

Configuración de Salida.

Un canal es usado para la salida, el Canal 0 es usado como salida para el control de voltaje a la entrada del amplificador PWM. El rango de la señal de salida es unipolar desde 0 a 10 voltios.

Conversiones D/A.

La fórmula siguiente calcula el voltaje de salida versus el código digital

$$V_{out} = 10.0\left(\frac{código\,digital}{4{,}096}\right) \qquad (43)$$

Puesto que el formato binario usado es binario directo, el código digital en la fórmula anterior es un valor decimal en el rango de 0 a 4095.

6.1.2 Hardware de Acondicionamiento de Señal.

A fin de leer valores de corriente de campo y velocidad de rotor desde la computadora, se construyo una interfase que adapta el bajo voltaje diferencial en la resistencia paralela, la cual sensa la corriente de campo, y adapta el bajo voltaje generado por el tacómetro, el cual es proporcional a la velocidad del motor.

Medida de la corriente de Campo.

El circuito mostrado en la Figura 18 se usa para medir la corriente de campo. Se usa una resistencia en paralelo Rs = 0.025 Ω en serie con el circuito de campo para sensar la caída de voltaje, la cual es proporcional a la corriente de campo. La salida del sensor esta dada por la ecuación siguiente

$$v_s = R_s\, i_F$$

El rango de esta caída de voltaje está en la orden de los milivoltios. A fin de amplificar el bajo voltaje de salida del sensor, se usa un amplificador DC de dos etapas. Debido a que la caída de voltaje del sensor es una salida diferencial, el amplificador debe desempeñar la función de

convertir la salida diferencial a salida de modo común referenciada a tierra, y es realizado por la primera etapa del amplificador.

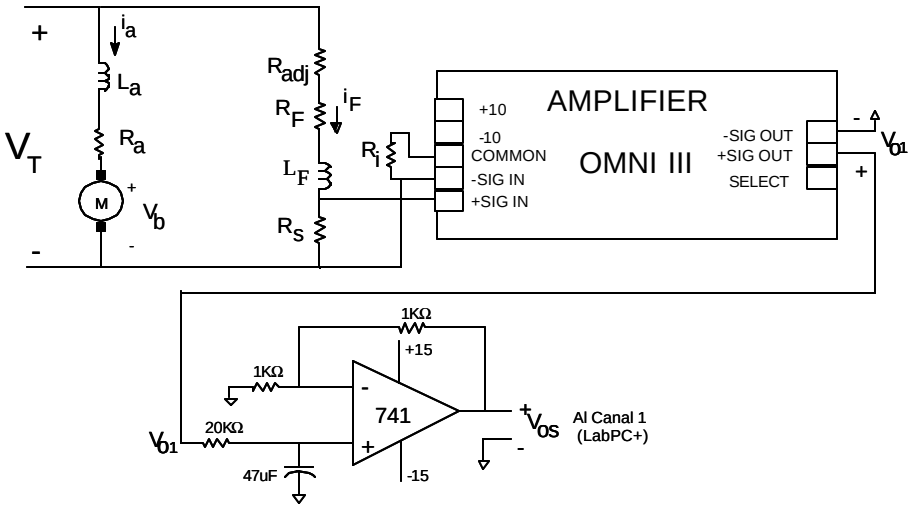


Figure 18: Medición de Corriente de Campo.

El amplificador de señal DC OMEGA'S OMNI-AMP^{TM} III es usado para la primera etapa del amplificador, y es un amplificador de instrumentación de propósito general. Para este amplificador, se usa una ganancia de 100 para entradas diferenciales. Para poder aislar la corriente del sensor desde la tierra Omni- Amp III conectando a (COMM), la entrada (-) se ha conectado al terminal COMM a través de una resistencia R_i como mostrado en la Figura 18. El valor de la resistencia es ajustada a 20 KΩ. Los terminales del sensor de resistencia paralela se conectan a los terminales de entradas no-inversoras del omni-amplificador.

Despreciando los términos pequeños de errores del OPAM, la ecuación para el voltaje de salida de la primera etapa, V_{01}, en términos del voltaje diferencial del resistor paralelo v_s es dada por

$$V_{01} = 100v_s$$

Un amplificador de Modulación por Ancho de Pulso (PWM) es usado para comandar el motor. Debido a la naturaleza de conmutación del PWM, hay una señal de alta frecuencia (25 K Hz en nuestro caso) que es el ruido que contiene la medición de la corriente de campo. Para eliminar este ruido, se ha agregado un filtro analógico pasa bajo, así es posible filtrar la señal antes del muestreo, y se ha realizado en la segunda etapa.

La salida de la segunda etapa del amplificador proporciona un filtro pasa bajo con la frecuencia de corte 0.17 Hz y una ganancia de 2. El voltaje de salida V_{os} de la segunda etapa en términos de la corriente de campo en amperios es dado aproximadamente por

$$V_{os} = 4.619 i_F \ \ voltios/\text{A} \qquad (44)$$

Medición de velocidad del Rotor.

La velocidad angular del rotor es medida por un tacómetro DC acoplada al eje del motor. El tacómetro empleado es un Kearfott Modelo # CR09610011 de Servo Systems con ganancia de 7V/1000RPM y con linealidad de 0.05%. Para eliminar la señal de alta frecuencia del ruido inducido por el amplificador PWM en la medición de velocidad, se considero una etapa de filtro similar a la usada para la medición de corriente de campo con frecuencia de corte de 0.17 Hz. Puesto que esta etapa proporciona una ganancia de 2 y el voltaje de entrada en el sistema de adquisición de datos es limitado a 10volts, se necesita atenuar el voltaje de salida del tacómetro antes de ingresarlo al circuito de filtro tal como se muestra en la Figura 19.

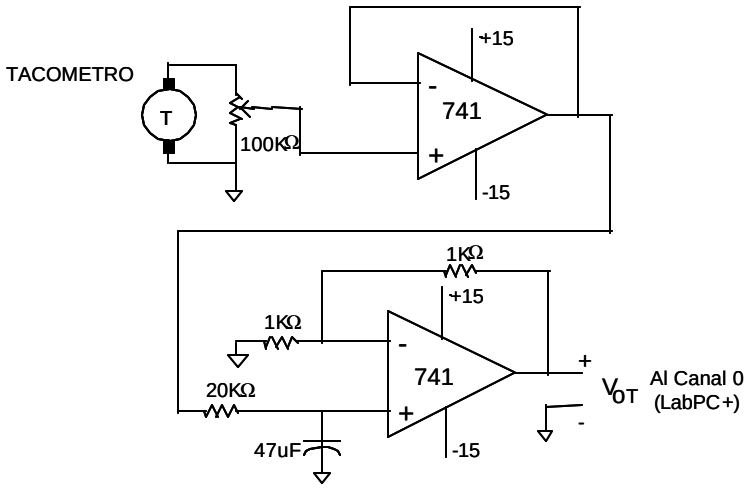


Figure 19: Medición de la Velocidad.

El voltaje de salida V_{ow} en voltios de la etapa de filtrado en términos de velocidad de rotor en rad/s es dado aproximadamente por

$$V_{0w} = 21.845 \times 10^{-3} w \quad voltios/rad/s \qquad (45)$$

6.1.3 Sistema de Potencia.

El amplificador para comandar el motor DC paralelo es el servo amplificador Copley 421 PWM, que proporciona una corriente continua de 5 Amp, un pico de 10Amp y una frecuencia de conmutación de 25 Khz. El servo amplificador se configura para trabajar como un amplificador de voltaje DC con ganancia de 10, como se muestra en la Figura 20. Puesto que el voltaje aplicado al motor es solo positivo, la referencia positiva de 0 a 10 voltios es usado, y el voltaje de salida del amplificador DC tiene un rango promedio de 0 a 100 Voltios. La señal referencia es proporcionado por el canal 0 de salida de la tarjeta de entrada/salida (E/S) Analógica y Digital LabPC+. La entrada de alimentación (potencia) no regulada al servo amplificador es de 135 Voltios DC@6Amps.

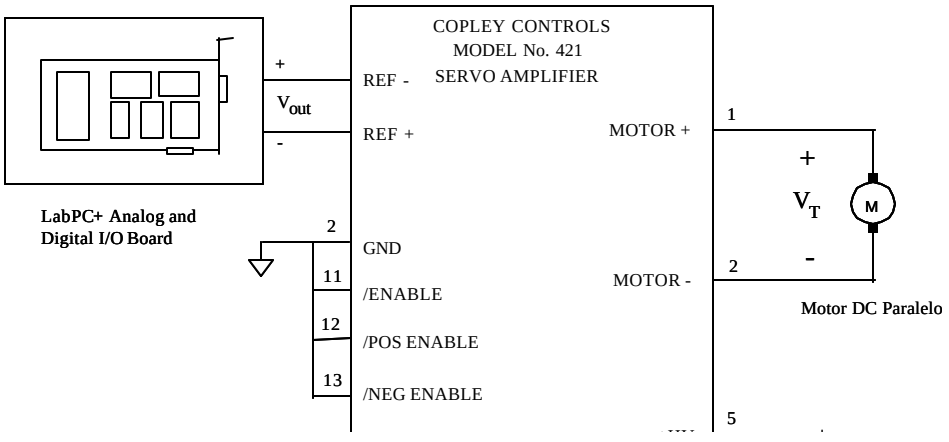


Figura 20: Configuración del Servo Amplificador como Amplificador de Voltaje DC

6.2 Uso del SIMULINK Real-Time Workshop.

El programa SIMULINK Real-Time Workshop™, es un generador automático de códigos en lenguaje C para SIMULINK. Produce códigos C de un modelo SIMULINK y automáticamente construye un programa que puede ejecutarse en tiempo-real.

Se usa las funciones-S para incorporar códigos C en programas de aplicación creados para trabajar en tiempo-real.

6.2.1 Ciclo de Desarrollo.

El modelo de controlador linealizante adaptivo fue diseñado, probado y analizado usando Matlab y SIMULINK. Después que esta etapa fue completada, el código C fue generado directamente del diagrama de bloques en SIMULINK, entonces el código C generado ha sido compilado y enlazado por el compilador Watcom C/C++ versión 10.0 (un compilador de 32-bits soportado para esta aplicación) para generar un archivo ejecutable del modelo del sistema.

Entonces, se ejecuta el código generado en tiempo real. Las variables de interés se registraron en un buffer circular por medio de opciones de registros de datos del SIMULIK Real-Time Workshop. Finalmente, los datos del programa se cargan en Matlab para el análisis y despliegue de los mismos. La Figura 21 muestra gráficamente este ciclo de desarrollo.

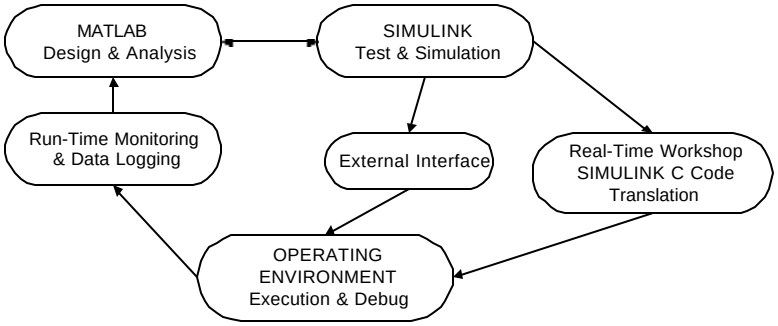


Figure 21: Ciclo de Desarrollo.

6.2.2 Generación Automática del Programa.

El proceso de hacer un programa de control autónomo llamado **adap_w.exe** desde el modelo SIMULINK **adap_w.m** usando Real-Time Workshop consiste de tres pasos principales:

1. La generación del código del modelo **adap_w.m.**
2. Generar un Makefile usando la plantilla Makefile **drt_watc.tmf.**
3. Invocar el utilitario de generación con el archivo **make_rt.m**.

La Figura 22 ilustra estos pasos. La Figura 23 ilustra la lógica que controla el proceso de construcción.

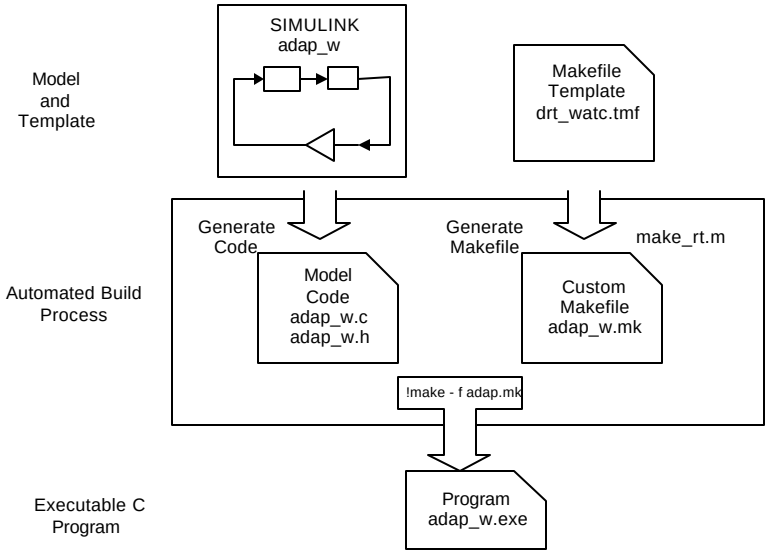


Figure 22: El Proceso de Implementación.

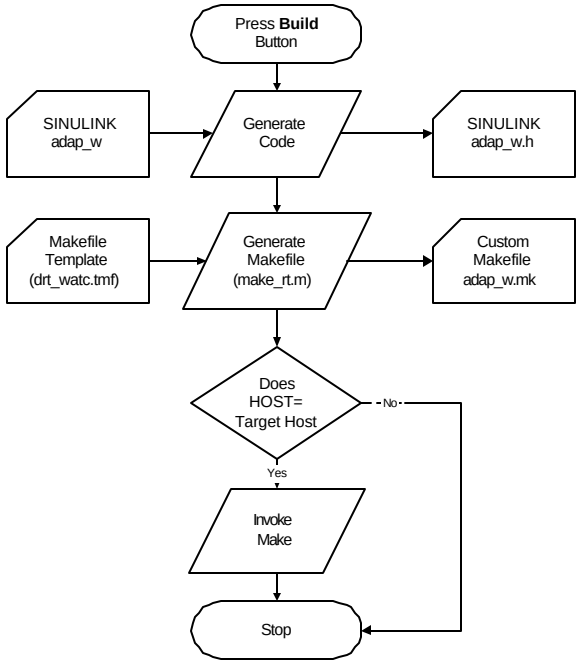


Figura 23: Elaboración del Programa de Lógica del Control Autónomo

6.2.3 Manejadores de Dispositivos.

Los bloques **labpc_ad** y **labpc_da** son los manejadores de dispositivos, implementados como códigos C de funciones-S **labpc_ad.c** y **labpc_da.c**, esto maneja la comunicación entre un programa en tiempo real y la tarjeta de adquisición de datos National Instruments LabPC+ I/O. Estas funciones de sistema interactúan recíprocamente con el SIMULINK durante la simulación y son enlazadas con el código generado. Estos bloques se enmascaran y tienen su propia caja de diálogo para ingresar comandos de inicialización, tal como la dirección Base de la tarjeta E/S, número de canales para la entrada ó salida, tiempo de muestreo y acceso a la habilitación del hardware. El código C que fue programado para estos dispositivos se incluye en el Apéndice A.

Las operaciones de estos manejadores de dispositivos con las funciones-S incluye:

1. Inicialización de *Simstruct.*
2. Inicialización al dispositivo E/S.
3. Cálculo de los bloques de salida según el tipo de manejador (*driver*) a ser implementado:
   Lectura de valores desde un dispositivo A/D LabPC+ y asignación de estos valores a los vectores de salida $\boldsymbol{y}$ del bloque.
   La escritura de valores del vector de entrada $\boldsymbol{u}$ del bloque a un D/A LabPC+.
4. Puesta a cero de la salida del DAC.

6.2.4 Temporización del Programa.

Para operar en tiempo real, el programa instala sus propias rutinas de servicio de interrupción (RSI) para ejecutar el código del modelo a intervalos predefinidos. El reloj del sistema de la computadora se uso para temporizar estos intervalos de muestreo. La selección de intervalo muestreo es muy importante. El intervalo de muestreo para esta aplicación se normalizó con respecto al tiempo de subida.

Considere N_i como el número de intervalos de muestreo por tiempo de subida, de la respuesta en lazo cerrado

$$N_i = \frac{T_r}{T}$$

donde T_r es el tiempo de subida. Es razonable escoger N_i entre 4 a 10 (Amstrom y Wittenmark, 1990). El tiempo de subida del sistema de lazo cerrado es aproximadamente 10 seg, eligiendo N_i = 10, entonces el periodo de muestreo como máximo debe ser *T = 1 seg.*

Puesto que el mayor intervalo de muestreo que puede definirse cuando se usa el contador/temporizador PC-AT's 8254 es aprox. 0.056 seg, entonces se eligió 0.04 seg para el intervalo de muestreo.

Los programas en Tiempo Real usan temporización por interrupciones para ejecutar la rutina del controlador. La Figura 24 ilustra la temporización de la interrupción.

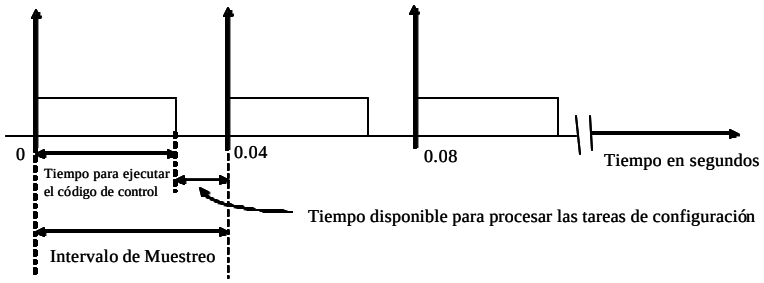


Figure 24: Temporización de Interrupción.

6.3 Programa en Tiempo real.

Usamos el mismo modelo en SIMULINK para el controlador linealizante adaptivo diseñado en el Capítulo 5 para construir un modelo del sistema en tiempo-real. En el programa en tiempo-real, el bloque **motor** se remueve del modelo de SIMULINK y se sustituye por la entrada y las salidas proporcionadas por los bloque **LabPC+ ADC** y **LabPC+ DAC** como se muestra en la Figura 25.

Cada bloque manejador de dispositivo ha sido configurado usando la caja de diálogo, lo siguiente muestra la configuración para cada bloque:

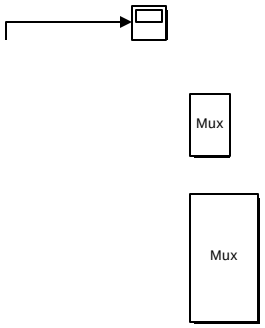


Figure 25 Controlador Adaptivo Linealizante, Implementación en Tiempo-Real.

- La **Dirección Base I/O** se configura a '0x260' para corresponder a la configuración de la tarjeta.
- Debido a que el bloque ADC lee dos entradas, el **Número de Canales** es configurado a 2. El bloque DAC escribe un canal de salida, así el parámetro **Número de Canales** es configurado a 1.
- El **Tiempo de Muestreo** de ambos bloques es configurado a 0.04 para corresponder al tamaño de paso del modelo. (Este valor es configurado con la caja de diálogos de la opción en tiempo real).

Las opciones siguientes muestran las configuraciones de las opciones de la caja de dialogo en tiempo real:

- El **Algoritmo** seleccionado es Heun, el algoritmo Heun de integración.
- El **Tamaño de Paso**, el intervalo de tiempo sobre el cual se ejecuta es configurado a 0.04 (un tamaño de paso de 0.04 significa que el programa actualiza sus salidas a una razón de 25 Hz).

- Las **Opciones de Registro de Datos** permite salvar el dato de Tiempo y de pantallas gráficas que corresponden a la velocidad del rotor, la salida del modelo referencia, la corriente de campo, la estimación del torque de carga, y voltaje de control de salida. (estas opciones se usan para el análisis y la presentación de los datos en MATLAB).

- La **Plantilla Makefile** usada para construir el programa en tiempo real es **drt_watc.tmf,** el cual fue configurado anteriormente de acuerdo al ambiente donde se ejecuta la aplicación. El ambiente usado para esta aplicación es un PC-AT corriendo en DOS, el Watcom C/C++ versión 10.0 Compiler, y el utilitario **wmake**.

Una vez que la estructura del proceso ha terminado, obtenemos la imagen ejecutable **adap_w.exe** (en adición a los archivos objeto creados en el proceso de construcción de los módulos fuentes).

El programa en tiempo-real **adap_w.exe** añade dos funciones al programa de simulación:

Primero, es capaz de leer y escribir datos desde el hardware externo (LabPC+ I/O Board). Los datos leídos son; la entrada de la medida de la corriente de campo y la velocidad del rotor, y el dato escrito es; el salida que controla la velocidad del motor.
Segundo, usa el dispositivo Contador/Temporizador PC-AT's 8254 para controlar la ejecución del programa, generando interrupciones con un intervalo de muestreo de 25 Hz (0.04 segundos).

Se usa el siguiente comando para ejecutar el programa en el intervalo de tiempo de 0 a 60 segundos.

Desde el prompt de Matlab:

> > **!adap_w 60**

Desde prompt DOS:

C: \ > adap_w 60

6.3.1 Consideraciones del Algoritmo de Integración

En las herramientas de simulación fuera de línea tal como SIMULINK, la simulación del modelo generalmente involucra la integración de conjuntos de ecuaciones diferenciales ordinarias, (tal como Runge-Kutta 45, Gear, Adams o Euler). El uso de métodos de orden elevado aumenta el tamaño de paso mínimo, debido a la mayor carga computacional. Normalmente no es recomendado este tipo de métodos, ya que el acceso a la tarjeta E/S se hará con menor frecuencia, generando problemas de ruido y traslape de señales en las entradas, y suavizando los problemas a las salidas más frecuentes.

Personas experimentadas trabajando con implementaciones en tiempo real (Hanselmann, 1993) sugieren que no es mala idea usar la simple integración de Euler. Otros métodos, tales como el método de segundo orden de Heun también pueden ser usados. El método de Heun tiene una región de estabilidad mayor que la de Euler, lo cual significa que un tamaño de paso mayor se puede usar con más exactitud si se obtiene para el mismo tamaño de paso.

6.4 Resultados Experimentales.

El sistema se ha probado con el motor DC paralelo al cual se le acopló mecánicamente un generador DC. La salida del generador entonces se aplicó a una carga resistiva variable. La carga resistiva se ajustó para proporcionar diferentes condiciones de carga. El desempeño de controlador se evaluó con las diferentes funciones de escalón como comando de velocidad.

6.4.1 Caso 1. Respuesta Críticamente Amortiguada.

En este caso el desempeño del controlador linealizante adaptivo se prueba usando un modelo de referencia de segundo orden críticamente amortiguado con $\zeta = 1$.

Comando de Velocidad escalón arriba (step-up)

La Figura 26 ilustra el desempeño de la aceleración del motor DC paralelo controlado por la linealización adaptiva por realimentación basado en modelo de referencia. Como podemos ver la velocidad real del rotor w sigue a la velocidad del modelo w_{ref} y por lo tanto se confirma que el controlador adaptivo propuesto tiene un buen desempeño. Los datos se han obtenido mediante de un sistema adquisición de datos, de la Figura 26 se puede ver que se obtiene un tiempo de establecimiento cerca de 20 segundos.

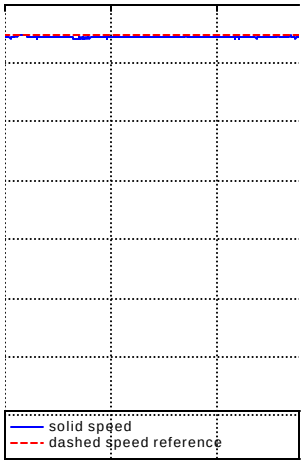


Figura 26: Respuesta de Velocidad(ζ = 1.0)

Con la finalidad de evaluar del controlador bajo torque de carga de perturbación, se uso una resistencia de 200 Ω conectada al generador DC para obtener una torque de carga t_L de aproximadamente de 0.15 N - m, esta perturbación de carga se aplicó en t = 25 seg, podemos ver a partir de la curva de velocidad, que la velocidad del rotor experimento una ligera caída, y el tiempo de recuperación de la velocidad del rotor estuvo sobre 3 segundos. Entonces, el controlador linealizante adaptivo de velocidad puede, por lo tanto, compensar la desviación de velocidad del rotor debido al torque de perturbación de carga.

La Figura 27 muestra el buen desempeño en estado estable del observador del torque de carga. La Figura 28 muestra el voltaje en los terminales del motor DC paralelo V_T a fin de lograr el seguimiento del modelo referencia.

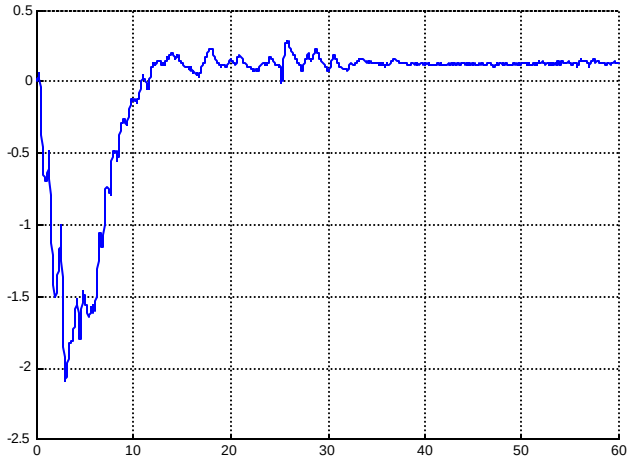


Figura 27: Estimación de torque carga (ζ = 1.0)

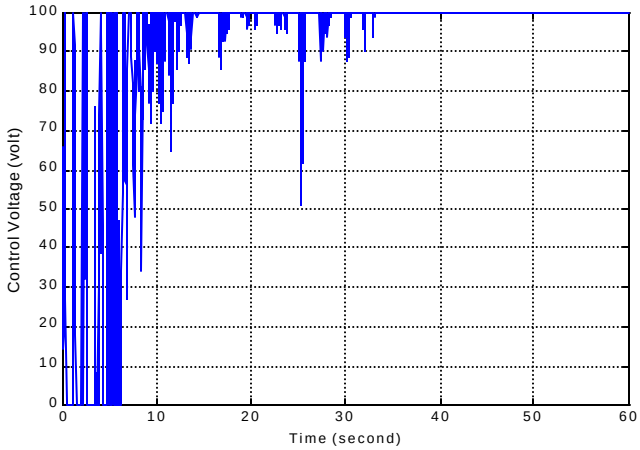


Figura 28: Voltaje de Control, V_T (ζ = 1.0)

Comando de Velocidad escalón arriba-abajo (step up-down).

La Figura 29 presenta el desempeño básico de aceleración del motor DC bajo el controlador de velocidad basado en linealización adaptiva, donde la línea entrecortada indica la trayectoria de referencia, la línea sólida corresponde a la velocidad del rotor del motor DC paralelo y la línea punteada representa el comando de velocidad.

La velocidad actual del rotor w sigue al modelo referencia w_r en respuesta al comando de velocidad *step up-down*, entonces se confirma que se ha realizado el seguimiento del modelo. Se obtuvo un tiempo de establecimiento de 15 seg aproximadamente para el comando step up-down.

El desempeño del controlador bajo el torque de carga de perturbación fue probado por medio de la resistencia de 200 Ω conectada al generador DC para obtener un torque de carga t_L aprox. de 0.15 *N-m*., esta perturbación de carga se aplicó en $t = 0$ seg.

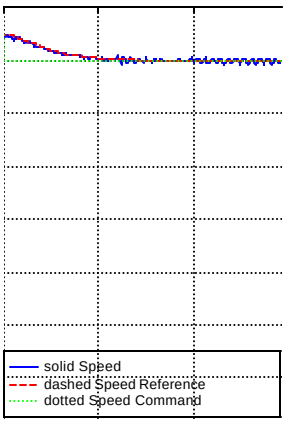


Figura 29: Respuesta de Velocidad ($\zeta = 1.0$)

La Figura 30 muestra el estimado de la perturbación de torque de carga. La Figura 31 muestra la señal de entrada de control V_T. La acción de control consiste en variar la amplitud del voltaje aplicado. El voltaje suministrado se limita a las capacidad del amplificador DC.

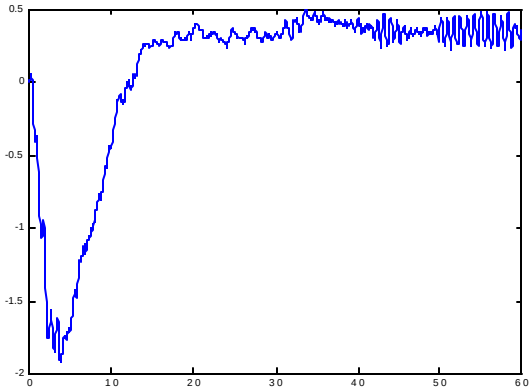


Figura 30: Estimación de torque de Carga(ζ = 1.0)

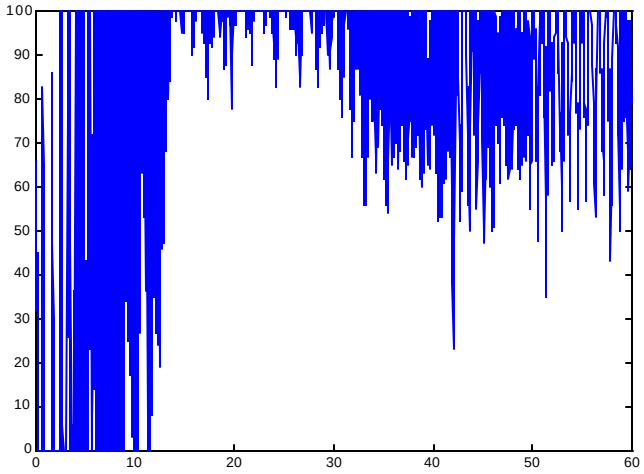


Figure 31: Voltaje de Control (ζ = 1.0)

Comando de Velocidad escalón Arriba-Abajo-Arriba (step-Up-Down-Up).

La Figura 32 muestra la respuesta de velocidad del rotor del motor DC paralelo, donde el comando velocidad son funciones de escalón.

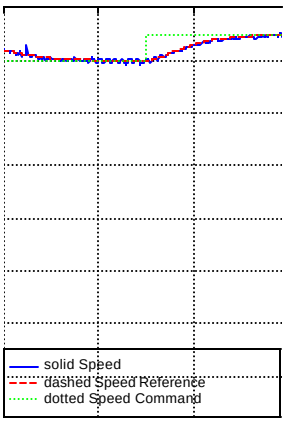


Figure 32: Respuesta de Velocidad (ζ = 1.0)

La curva de respuesta claramente muestra que la velocidad medida sigue el modelo referencia muy cercanamente, donde la línea entrecortada indica la trayectoria de referencia,

la línea sólida corresponde a la velocidad del rotor y la línea punteada representa el comando de velocidad.

También en este caso, el desempeño del controlador bajo torque de carga de perturbación fue probada por medio del generador DC cargado con un resistor de 200 Ω conectado al eje del motor DC paralelo, en esta condición un torque de carga t_L aproximadamente de 0.15 N - m. se aplicó en t = 0 seg.

La Figura 33 muestra la estimación de la perturbación de torque de carga. La Figura 34 muestra el control de voltaje V_T aplicado al terminal de entrada del motor DC paralelo a fin de lograr un seguimiento del modelo referencia.

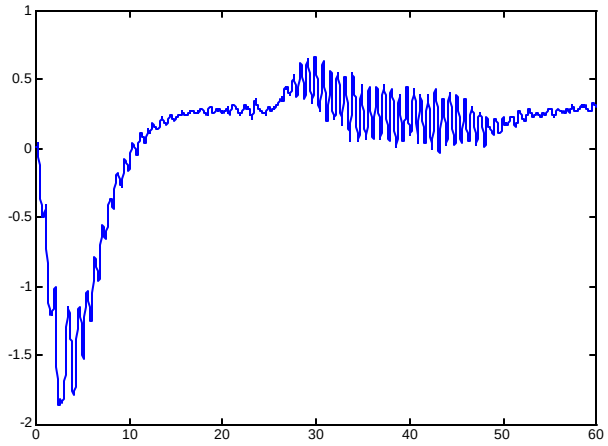


Figura 33: Estimación de torque de carga (ζ=1.0)

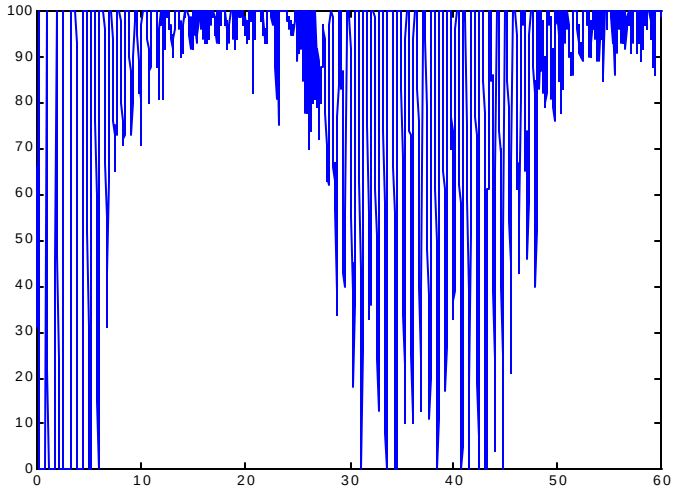


Figura 34: Voltaje de Control (ζ = 1.0)

6.4.2 Caso 2. Respuesta Sub-Amortiguada.

Para este caso el desempeño del controlador linealizante adaptivo fue probado usando un modelo referencia de segundo orden subamortiguado con ζ = 0.707.

Comando de Velocidad escalón Arriba-Abajo-Arriba (step-Up-Down-Up).

La Figura 35 muestra la medida experimental de velocidad del motor DC paralelo. Observe los efectos sobre la respuesta transitoria inicial de velocidad del motor debido a la saturación del amplificador DC. El error de seguimiento aumenta durante la saturación, pero el sistema se recupera después de aproximadamente 10 seg y recobra el seguimiento de la trayectoria de referencia. También, podemos ver que para cambios pequeños en el comando de velocidad, la velocidad del motor sigue al modelo de referencia. El desempeño de controlador bajo torque de carga de perturbación fue probada usando las mismas condiciones de carga usadas en el Caso 1.

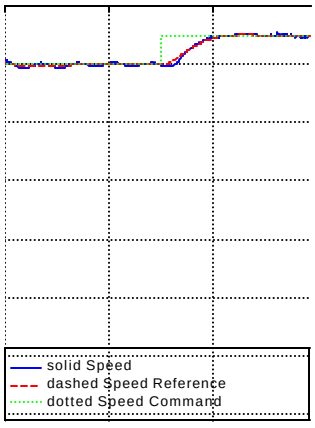


Figure 35: Respuesta de Velocidad (ζ= 0.707)

Si el comando del amplificador del motor excede a 100 voltios, entonces ocurren los siguientes sucesos:

(1) el comando se limita a 100volts,

(2) el retraso de la velocidad del motor al modelo de referencia, y

(3) el aumento de error de velocidad.

La Figura 36 muestra el voltaje de control aplicado a los terminales del motor DC paralelo. Observe la saturación de voltaje a la salida del amplificador DC a 100 voltios, ocasionando la degradación de la respuesta transitoria inicial de velocidad.

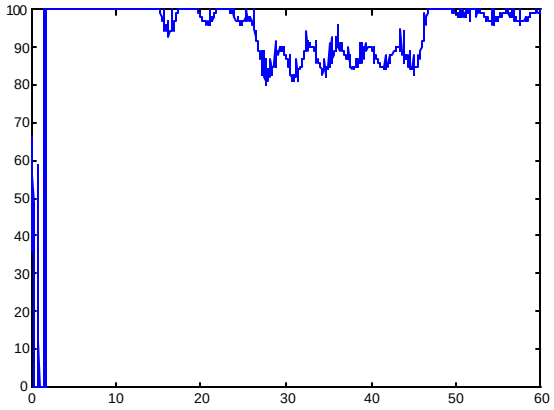


Figura 36: Voltaje de Control ($\zeta = 0.707$)

6.4.3. Caso 3. Desempeño de Estado Estable.

En este caso se aplica una carga $(t_L \neq 0)$ al motor que esta en movimiento por un periodo largo sin carga $(t_L = 0)$. Los valores registrados en ambas condiciones son:

Para $t_L = 0$:

Velocidad del Rotor = 149.95 *rad/seg*.

Voltaje de Salida del Amplificador = 94.5*volts*.

Potencia de entrada en el motor = 68.371*Watts*

Torque Electromagnético = 0.456*N - m*

Para $t_L = 0.15N - m$:

Velocidad del Rotor = 149.95 *rad/seg*

Voltaje de Salida del Amplificador = 97.2*volts*

Potencia de entrada en el motor = 82.08*Watts*

Torque Electromagnético = 0.547*N - m*

Potencia de Salida del generador DC = 11.14*Watts*

Observe que el valor final de la velocidad bajo condiciones de carga y sin carga son iguales, también observe la variación del voltaje de salida del amplificador a fin de compensar la caída en velocidad cuando se aplica la carga.

6.4.4 Conclusiones de la Pruebas Experimentales.

El resultado experimental en el Caso 1 muestra que la respuesta de velocidad del motor efectivamente sigue al modelo referencia bajo la perturbación de carga y bajo la variación en la velocidad de mando (Step-Up y Step-Down), se puede ver también el buen desempeño del algoritmo usado para estimar el torque de carga. En el Caso 2, un modelo de referencia subamortiguado se usó a fin de ver la degradación de la respuesta de velocidad debido a la saturación de amplificador DC, que es corroborada por las gráficas de la Figura 36.

Los resultados de Caso 3 demuestran que el desempeño dinámico del controlador linealizante adaptivo es satisfactorio, no se presenta error en estado estable después de la recuperación del cambio de carga. Los resultados antes mencionados demuestran que el algoritmo de control linealizante adaptivo trabaja muy bien cuando se implementa en tiempo-real.

Debido a las restricciones de hardware en el montaje experimental no fue posible evaluar el algoritmo adaptivo para la velocidad por encima de 150*rad/seg* y el uso del comando de escalón de velocidad con un rango mas amplio.

Capítulo 7

Conclusiones, Comentarios, Trabajo Futuro y Costos.

Se ha mostrado como teorías de control no lineal tal como linealización por realimentación, puede aplicarse efectivamente para controlar motores eléctricos DC. El desarrollo de la propuesta del controlador linealizante adaptivo para mejorar el controlador linealizante frente a parámetros desconocidos en el sistema ha sido implementado experimentalmente en forma exitosa.

El observador no-lineal con dinámica de error lineal para estimar el torque de carga se implementó para proveer una estimación del torque de carga al controlador adaptivo.

Los resultados del diseño analítico han sido verificados: primero por simulación y luego por una implementación experimental. Los resultados experimentales han demostrado que el controlador linealizante adaptivo combinado con el observador permite mantener el desempeño prescrito de control ante la presencia de parámetros desconocidos en el sistema controlado y el torque de carga de perturbación.

Los resultados de la simulación y los datos experimentales están en correspondencia cuando no existe saturación en el amplificador DC, y el control de velocidad realizado funciona muy bien. La estructuras del control sugeridas; método de control de ajuste de parámetros y las leyes de control no lineal son de aplicación general; estos también se aplica al control de posición.

La dinámica adaptiva del controlador y la dinámica del observador son válidas si la corriente de campo es mayor de cero, puesto que esta condición representa una operación normal del motor DC paralelo, ello se satisface para cualquier punto de operación en el rango de funcionamiento del motor.

Existe un problema en implementar este controlador adaptivo en tiempo-real, este fue la singularidad en el algoritmo de control cuando la corriente de campo es cero. Para resolver este problema, la corriente de campo en el controlador y el torque de carga observada fue limitada a 0.01 A. Esta restricción no degradó el desempeño del controlador.

La metodología seguida para el diseño, prueba, análisis, y el proceso de hacer un programa autónomo que se ejecute en tiempo real usando Matlab y SIMULINK Real-Time Workshop ™ se ha descrito en forma detallada en esta tesis.

El uso potencial del SIMULINK Real-Time Workshop ™ en construir un programa que corra en tiempo real es sumamente beneficioso si se hace del modo correcto. La tecnología de hoy hace la hace accesible para todos como una técnica estándar.

El modelo continuo del controlador linealizante adaptivo se ha incorporado en el código C generado por SIMULINK™ usando el algoritmo de Heun de paso-fijo. Los beneficios de usar tales herramientas para construir un programa de tiempo real son menos tiempo en el desarrollo de la aplicación, mayor confiabilidad y bajo costo.

7.1 Trabajo Futuro.

Controlador de velocidad sin sensor de velocidad:

El desempeño del controlador cuando no se dispone de la medición de la velocidad debe ser considerado, tal observador basado en el modelo del motor, podría proporcionar la confiabilidad al sistema existente si se presenta una falla del sensor de velocidad o la necesidad de usar este motor sin el sensor de velocidad.

La posibilidad de desarrollar un observador de la velocidad rotor para un motor DC paralelo con dinámica de error linealizable fue investigado. Tal algoritmo de observador es basado en el trabajo previo de (Krener e Isidori, 1983) (Krener y Respondek, 1985) (Xia y

Gao, 1989). Debido a que el modelo de segundo orden usado para el control adaptivo de velocidad no es observable con la velocidad como salida, se ha aumentado otro estado que representa la dinámica de la corriente de armadura, este sistema de tercer orden es observable con la velocidad como salida. Las condiciones impuestas sobre el sistema para la existencia de tal observador es que se debería transformar en la forma canónica de observador no lineal.

El sistema de tercer de orden aunque es observable, no satisface la proposición 3.3 de (Krener y Respondek, 1985), no es posible de resolver ecuaciones diferenciales parciales no lineales a fin de obtener una transformación espacio-estado que transforma el sistema en una forma canónica observable. Entonces el observador de velocidad no puede ser construido por este enfoque.

Existe otro enfoque opuesto a todos los métodos conocidos de diseño de observador no lineal basado en forma canónica, la aplicación de estas reglas no requiere integración de ecuaciones diferenciales, ni una inversión de ecuaciones no lineales. La única consideración es que el sistema no lineal tal como un motor DC paralelo es observable, ver (Zeitz, 1987) y (Birk y Zeitz, 1988) para sistemas de entrada-salida multivariable.

Las restricciones con esta técnica son debidas a la presunción de triple diferenciación de funciones de sistemas no lineales y las derivadas de tercer orden de la entrada, estas derivadas pueden dificultar la medida exacta y puede añadir ruido al sistema.

Existen otras técnicas que deberían ser investigadas a fin de lograr un buen desempeño del observador de velocidad del rotor, ver (Misawa y Hedrick, 1989) para la revisión del estado del arte de observadores no lineales, una de esta técnica puede usarse para el desarrollo de un observador de velocidad para un motor DC paralelo.

Interfase Hombre - Máquina:

Crear una interfase de hombre-maquina para mostrar el menú, recepción de comandos, reporte de estado y visualización de los datos. El controlador adaptivo debería configurarse para reconocer fallas y capacidad de manejo cuando una de las siguientes condiciones existe: detección de cortocircuito, detección de sobre corriente, límite de corriente y límite de velocidad. Otras opciones que deberán ser añadidas son el permitir al usuario aumentar y disminuir la velocidad de referencia por teclado, mientras el motor esta en movimiento.

Ley de control:

Mejorar la ley de control linealizante adaptivo a fin de evitar singularidades.

7.2 Costo de Implementación

El costo de implementación del hardware, software y costo de ingeniería para el desarrollo del controlador adaptivo para un motor DC paralelo empleando linealización, en tiempo real usando Matlab y SIMULINK Real-Time Workshop ™ se muestra en la Tabla 5.

Tabla 5. Costo de Implementación

Modulo de Medición de Corriente				
Item	Descripción	Cantidad	Costo Unitario U.S. $	Total
1	Amplificador de Señal DC Omni III	1	336	336
2	Amplificador Operacional	1	2	2
3	Resistencia shunt	1	1	1
				339

Modulo de Medición de Velocidad				
Item	Descripción	Cantidad	Costo Unitario U.S. $	Total
4	Amplificador Operacional	1	2	2
5	Tacómetro	1	130	130
				132

Modulo Motor y Apmplificación de Voltaje				
Item	Descripción	Cantidad	Costo Unitario U.S. $	Total
6	Motor	1	1000	1000
7	Copley 421	1	500	500
				1500

Tarjeta de Adquisición de Datos				
Item	Descripción	Cantidad	Costo Unitario U.S. $	Total
8	Lab PC+ y bornera de conexión	1	1000	1000
				1000
		Costo del Hardware		**2971**

Software de Simulación y Control				
Item	Descripción	Cantidad	Costo Unitario U.S. $	Total
9	Matlab	1	2,500	2500
10	Simulink Real Time Workshop	1	250	250
11	Control Toolbox	3	250	750
		Costo del Software		**3500**

Costo de Ingeneiría				
Item	Descripción	Meses	Costo Mensual U.S. $	Total
12	Remuneración	4	500	2000
				2000
		Costo Total U.S. $		**8471**

Referencias

Åmström K.J. and B. Wittenmark. 1995. Adaptive Control. Addison-Wesley Publishing Company, Massachusetts, U.S.A., 1 pp.

Åmström K.J. and B. Wittenmark. 1990. Adaptive Control. Prentice-Hall, New Jersey, U.S.A., 1 pp.

Baumann W. T. and W. J. Rugh. 1986. Feedback control of nonlinear system by extend linearization. IEEE Transactions on Automatic Control. 31(1): 40-46.

Birk J. and M. Zeitz. 1988. Extended Luenberger observer for non-linear multivariable systems. International Journal Control. 47(6):1823-1836.

Bodson M., J. Chiasson. 1993. Nonlinear control of a shunt DC motor. IEEE Transactions on Automatic Control. (38)11:1662-1666.

Charlet B. and J. Lévine94. On dynamic feedback linearization. Systems & Control Letters. 13:143-151.

Chiasson J.. 1994. Nonlinear differential-geometric techniques for control of a series DC motors. IEEE Transactions on Automatic Control. 2(1):35-42.

Chiasson J. and M. Bodson. 1991. Nonlinear and Adaptive Control of a Shunt DC motor. Proceedings of the IEEE International Conference on Systems Engineering. pp. 73-76.

Fitzgerald A. E., C. Kingsley, and S. D. Umans. 1983. Electric Machinery. McGraw-Hill Book Company, New York, U.S.A., xxx pp.

Hanselmann H. 1993. Hardware-in-Loop Simulation as a Standard Approach for Development, Customization, and Production Tets of ECU's. Society of Automotive Engineers. paper 931953.

Hirschorn R. M.. 1990. Global controllability of locally linearizable systems. Siam J. Control and Optimization. 28(3):540-554.

Hunt L. R. H. , R. Su, and G. Meyer. 1983. Global transformations of nonlinear systems. IEEE Transactions on Automatic Control. 28(1):24-31.

Isidori A.. 1989. Nonlinear Control Systems. Springer-Verlag, New York, U.S.A., pp. 145-233.

Isidori A. and A. Ruberti. 1984. On the synthesis of linear input-output responses for nonlinear systems. Systems & Control Letters. 4:17-22.

Jakubczyk B. and W. Respondek. 1980. On linearization of control systems. Bulletin de L'académie Polonaise des Sciences. Séries des sciences mathématiques. 28:517-522.

Kailath T..1980. Linear Systems. Prentice Hall, New Jersey, U.S.A., 262-263 pp.

Krause P.C..1983. Analysis of Electrical Machines. McGraw-Hill, New York, U.S.A., pp. 73-94.

Krener A. J. and A. Isidori. 1983. Linearization by output injection and nonlinear observers. Systems & Control Letters. 3:47-52.

Krener A. J. and W. Respondek. 1985. Nonlinear observer with linearizable errors dynamics. SIAM J. Control and Optimization. 23(2):197-216.

Leonhard W.. 1985. Control of Electrical Drives. Springer-Verlag, Heidelberg, Germany, pp. 41-56.

Misawa E.A. and Hedrick J.K. 1989. Nonlinear observers - a state of the art survey. Journal of Dynamic Systems, Measurement and Control, 111:344—352.

McPherson G and R.D. Laramore. 1990. An Introduction to Electrical Machines and Transformers. John Wiley & Sons, New York, U.S.A., 349 pp.

Oliver P. D.. 1991. Feedback linearization of DC motors. IEEE Transactions on Industrial Electronics. 38(6):498-501.

Sastry S.S. and M. Bodson. 1989. Adaptive Control: stability, convergence and robustness. Prentice-Hall, New Jersey, U.S.A. , pp. 294-311.

Sastry S. S. and A. Isidori. 1989. Adaptive control of linearizable systems. IEEE Transactions on Automatic Control. 34(11):1123-1131.

Slotine J. J. E. and W. Li. 1991. Applied Nonlinear Control. Prentice-Hall, New Jersey, U.S.A. , pp. 229-235.

Venkatesan K. and P. Mukhopadhyay. 1972. Transfer Functions of D.C. Motors. Journal of the Institution of Engineer. 52(10):279-285.

Xia X. and W. Gao. 1989. Nonlinear observer design by observer error linearization. SIAM J. Control and Optimization. 27(1):199-216.

Zeitz M.. 1987. The extended Luemberger observer for nonlinear systems. Systems & Control Letters. 9:149-156.

Apéndice A
Bloques de Funciones-S

Lo siguiente muestra la implementación en código C como funciones-S, de los bloques del modelo usado para la implementación del programa en Tiempo-Real.

Modelo del Motor DC Paralelo

```
/*
 * MOTOR        Motor Model
 *
 *              Syntax  [sys, x0] = motor(t, x, u, flag)
 */

/*
 * The following #define is used to specify the name of this S-Function.
 */

#define S_FUNCTION_NAME motor
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif
/*
 *DC shunt motor nominal parameters
 *B=0.0011
 *J=0.1
 *Kdc=2.5
 *Ra=15.9
 *Radj=147
 *Rf=267
 *Lf=24.736
 *La=18.7452e-03;
 *N1=B/J;N2=(Kdc^2)/(J*Ra);N4=Kdc/(J*Ra);N3=(Rf+Radj)/Lf;N5=1/Lf;
 */
#define   N1        0.011
#define   N2        3.9308
#define   N3        16.7367
#define   N4        1.5723
#define   N5        0.04043
#define   J         0.1
/*
```

```
 * mdlInitializeSizes - initialize the sizes array
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */


static void mdlInitializeSizes(S)
   SimStruct *S;
{
   ssSetNumContStates(    S, 2);      /* number of continuous states */
   ssSetNumDiscStates(    S, 0);      /* number of discrete states */
   ssSetNumInputs(        S, 2);      /* number of inputs */
   ssSetNumOutputs(       S, 2);      /* number of outputs */
   ssSetDirectFeedThrough(S, 1);      /* direct feedthrough flag */
   ssSetNumSampleTimes(   S, 1);      /* number of sample times */
   ssSetNumInputArgs(     S, 0);      /* number of input arguments */
   ssSetNumRWork(         S, 0);      /* number of real work vector elements */
   ssSetNumIWork(         S, 0);      /* number of integer work vector elements */
   ssSetNumPWork(         S, 0);      /* number of pointer work vector elements */
}

/*
 * mdlInitializeSamp leTimes - initialize the sample times array
 * Since the S-functions is continuous a sample time of 0.0 was specified.
*/

static void mdlInitializeSampleTimes(S)
   SimStruct *S;
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 * This function is used to initialize the continuous states of  the S-function block
 * The initial states are placed in the x0 variable
 */

static void mdlInitializeConditions(x0, S)
   double *x0;
   SimStruct *S;
{
   x0[0] = 0.0;
   x0[1] = 0.0;
}

/*
 * mdlOutputs - compute the outputs
 * This  function is used to compute the outputs of the S-function block.
 * The outputs are placed in the y variable.
 */
```

```
static void mdlOutputs(y, x, u, S, tid)
   double *y;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
            y[0] = x[0];
            y[1] = x[1];
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */

static void mdlUpdate(x, u, S, tid)
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
}

/*
 * mdlDerivatives - compute the derivatives
 * This function is used to compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(dx, x, u, S, tid)
   double *dx;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
            dx[0]= -N1*x[0]-N2*x[0]*(x[1]*x[1])+N4*x[1]*u[0]-(1/J)*u[1];
            dx[1]= -N3*x[1]+N5*u[0];
}

/*
 * mdlTerminate - called when the simulation is terminated.
*/

static void mdlTerminate(S)
   SimStruct *S;
{
}
```

```
#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Model de Referencia

```
/*
 * REF    Reference Model
 *
 *                      Syntax  [sys, x0] = ref(t, x, u, flag)
 */

/*
 * The following #define is used to specify the name of this S-Function.
 */

#define S_FUNCTION_NAME ref
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif

/*
*Second order system reference model
*xmdot=Am*xm + Bm*u
*ym=cm'*xm
*Am=[0 1;-b -a];bm=[0 kp]; cm'=[1 0];
* For underdamped reference model, the damping_ratio = 0.707 was used
* For overdamped reference model, the damping_ratio = 1was used
* The settling time used for both conditions was 10 seconds
* DC_gain = 1.5
*wn=4/(settling_time*damping_ratio)
*a=2*damping_ratio*wn
*b=wn^2
*kp=DC_gain*b
*/
#define    b          0.32
#define    kp         0.48
#define    a          0.8
/*
 * mdlInitializeSizes - initialize the sizes array
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */

static void mdlInitializeSizes(S)
   SimStruct *S;
```

```
{
   ssSetNumContStates(    S, 2);      /* number of continuous states */
   ssSetNumDiscStates(    S, 0);      /* number of discrete states */
   ssSetNumInputs(        S, 1);      /* number of inputs */
   ssSetNumOutputs(       S, 2);      /* number of outputs */
   ssSetDirectFeedThrough(S, 1);      /* direct feedthrough flag */
   ssSetNumSampleTimes(   S, 1);      /* number of sample times */
   ssSetNumInputArgs(     S, 0);      /* number of input arguments */
   ssSetNumRWork(         S, 0);      /* number of real work vector elements */
   ssSetNumIWork(         S, 0);      /* number of integer work vector elements */
   ssSetNumPWork(         S, 0);      /* number of pointer work vector elements */
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 * Since the S-functions is continuous a sample time of 0.0 was specified.
*/

static void mdlInitializeSampleTimes(S)
   SimStruct *S;
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 * This function is used to initialize the continuous states of  the S-function block
 * The initial states are placed in the x0 variable
 */

static void mdlInitializeConditions(x0, S)
   double *x0;
   SimStruct *S;
{
   x0[0] = 0.0;
   x0[1] = 0.0;
}

/*
 * mdlOutputs - compute the outputs
 * This  function is used to compute the outputs of the S-function block.
 * The outputs are placed in the y variable.
 */



static void mdlOutputs(y, x, u, S, tid)
   double *y;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
```

```
{
        y[0] = x[0];
        y[1] = x[1];
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */

static void mdlUpdate(x, u, S, tid)
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
}

/*
 * mdlDerivatives - compute the derivatives
 * This function is used to compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(dx, x, u, S, tid)
   double *dx;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
        dx[0]= x[1];
        dx[1]= -b*x[0]-a*x[1]+kp*u[0];
 }

/*
 * mdlTerminate - called when the simulation is terminated.
*/


static void mdlTerminate(S)
   SimStruct *S;
{
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
```

```
#endif
```

Observador del Torque de Carga

```
/*
 * TQ     Load Torque Observer Model
 *
 *                      Syntax  [sys, x0] = tq(t, x, u, flag)
 */

/*
 * The following #define is used to specify the name of this S-Function.
 */

#define S_FUNCTION_NAME tq
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif
/*
 *Load Torque Observer
 *x1=speed estimate; x2=field current estimate; x3=load torque estimate/J
 *DC shunt motor nominal parameters
 *B=0.0011
 *J=0.1
 *Kdc=2.5
 *Ra=15.9
 *Radj=147
 *Rf=267
 *Lf=24.736
 *La=18.7452e-03;
 *N1=B/J;N2=(Kdc^2)/(J*Ra);N4=Kdc/(J*Ra);N3=(Rf+Radj)/Lf;N5=1/Lf;
 */
#define    N1          0.011
#define    N2          3.9308
#define    N3          16.7367
#define    N4          1.5723
#define    N5          0.04043
#define    J           0.1
/*
 *Matrix Gain L for the error dynamics of the observer
 */
#define    l11         10
#define    l12         -10
#define    l21         10
#define    l22         10
#define    l31         -10
#define    l32         10
/*
 * mdlInitializeSizes - initialize the sizes array
```

```
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */

static void mdlInitializeSizes(S)
   SimStruct *S;
{
   ssSetNumContStates(    S, 3);      /* number of continuous states */
   ssSetNumDiscStates(    S, 0);      /* number of discrete states */
   ssSetNumInputs(        S, 3);      /* number of inputs */
   ssSetNumOutputs(       S, 1);      /* number of outputs */
   ssSetDirectFeedThrough(S, 1);      /* direct feedthrough flag */
   ssSetNumSampleTimes(   S, 1);      /* number of sample times */
   ssSetNumInputArgs(     S, 0);      /* number of input arguments */
   ssSetNumRWork(         S, 0);      /* number of real work vector elements */
   ssSetNumIWork(         S, 0);      /* number of integer work vector elements */
   ssSetNumPWork(         S, 0);      /* number of pointer work vector elements */
}

/*
 * mdlInitializeSampleTimes - initialize the sample times array
 * Since the S-functions is continuous a sample time of 0.0 was specified.
*/

static void mdlInitializeSampleTimes(S)
   SimStruct *S;
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 * This function is used to initialize the continuous states of  the S-function block
 * The initial states are placed in the x0 variable
 */


static void mdlInitializeConditions(x0, S)
   double *x0;
   SimStruct *S;
{
   x0[0] = 0.1;
   x0[1] = 0.1;
   x0[2] = 0.1;
}

/*
 * mdlOutputs - compute the outputs
 * This  function is used to compute the outputs of the S-function block.
 * The outputs are placed in the y variable.
 */
```

```
static void mdlOutputs(y, x, u, S, tid)
   double *y;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
        y[0] = x[2];
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */

static void mdlUpdate(x, u, S, tid)
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
}

/*
 * mdlDerivatives - compute the derivatives
 * This function is used to compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(dx, x, u, S, tid)
   double *dx;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
        dx[0]= -N1*x[0]-N2*x[0]*(x[1]*x[1])+N4*x[1]*u[0]-(1/J)*u[1];
        dx[1]= -N3*x[1]+N5*u[0];
        if (u[1]<0.001)
        {
                u[1]=0.001;
                dx[0]=-x[2]-N1*u[0]-N2*u[0]*(u[1]*u[1])+N4*u[1]*u[2]+l11*(u[0]-x[0])+
                        l12*(u[1]-x[1]);
                dx[1]=-N3*u[1]+N5*u[2]+l21*(u[0]-x[0])+l22*(u[1]-x[1]);
                dx[2]=l31*(u[0]-x[0])+l32*(u[1]-x[1]);
        }
        else
        {
                dx[0]=-x[2]-N1*u[0]-N2*u[0]*(u[1]*u[1])+N4*u[1]*u[2]+l11*(u[0]-x[0])+
```

```
                    l12*(u[1]-x[1]);
                dx[1]=-N3*u[1]+N5*u[2]+l21*(u[0]-x[0])+l22*(u[1]-x[1]);
                dx[2]=l31*(u[0]-x[0])+l32*(u[1]-x[1]);
        }
}

/*
 * mdlTerminate - called when the simulation is terminated.
*/

static void mdlTerminate(S)
   SimStruct *S;
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Ley de Control Linealizante Adaptivo

```
/*
 *LW      Adaptive Feedback Linearization Law Model
 *
 *                  Syntax  [sys, x0] = lw(t, x, u, flag)
*/

/*
 * The following #define is used to specify the name of this S-Function.
 */

#define S_FUNCTION_NAME lw
/*
 * need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif
/*
 *nominal parameters of DC shunt motor
 *B=0.0011;J=0.1;Kdc=2.5;Ra=15.9;Radj=147;Rf=267;Lf=24.736;La=18.7452e-03;
 *N1=B/J;N2=(Kdc^2)/(J*Ra);N4=Kdc/(J*Ra);N3=(Rf+Radj)/Lf;N5=1/Lf; N6= x3 = load_torque/J
 */
#define    N1        0.011
#define    N2        3.9308
#define    N3        16.7367
#define    N4        1.5723
#define    N5        0.04043
#define    alpha     10
```

```
/*
 * mdlInitializeSizes - initialize the sizes array
 * The sizes array is used by SIMULINK to determine the S-function block's
 * characteristics (number of inputs, outputs, states, etc.).
 */

static void mdlInitializeSizes(S)
   SimStruct *S;
{
   ssSetNumContStates(    S, 7);      /* number of continuous states */
   ssSetNumDiscStates(    S, 0);      /* number of discrete states */
   ssSetNumInputs(        S, 5);      /* number of inputs */
   ssSetNumOutputs(       S, 3);      /* number of outputs */
   ssSetDirectFeedThrough(S, 1);      /* direct feedthrough flag */
   ssSetNumSampleTimes(   S, 1);      /* number of sample times */
   ssSetNumInputArgs(     S, 0);      /* number of input arguments */
   ssSetNumRWork(         S, 0);      /* number of real work vector elements */
   ssSetNumIWork(         S, 0);      /* number of integer work vector elements */
   ssSetNumPWork(         S, 0);      /* number of pointer work vector elements */
}
/*
 * mdlInitializeSampleTimes - initialize the sample times array
 * Since the S-functions is continuous a sample time of 0.0 was specified.
*/

static void mdlInitializeSampleTimes(S)
   SimStruct *S;
{
   ssSetSampleTimeEvent(S, 0, 0.0);
   ssSetOffsetTimeEvent(S, 0, 0.0);
}

/*
 * mdlInitializeConditions - initialize the states
 * This function is used to initialize the continuous states of  the S-function block
 * The initial states are placed in the x0 variable
 */

static void mdlInitializeConditions(x0, S)
   double *x0;
   SimStruct *S;
{
   x0[0] = 0.0;
   x0[1] = 0.01;
   x0[2] = 3.9;
   x0[3] = 16.5;
   x0[4] = 1.5;
   x0[5] = 0.04;
   x0[6] = 0.1;
}

/*
 * mdlOutputs - compute the outputs
```

```
 * This  function is used to compute the outputs of the S-function block.
 * The outputs are placed in the y variable.
 */

static void mdlOutputs(y, x, u, S, tid)
   double *y;
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
            int u1;
            int v;
            if (u[1]<0.001) {
                        u[1]=0.001;
            }
            v = u[4]+alpha*(u[3]-u[0]);
            u1=(1/(x[4]*u[1]))*(-(-x[1]*u[0]-x[2]*u[0]*(u[1]*u[1])-x[6])+(v));
            if (u1>100.0) {
                        u1 = 100.0;
            }
            if (u1<0) {
                        u1 = 0;
            }
                        y[0] = u1;
}

/*
 * mdlUpdate - perform action at major integration time step
 *
 * This function is called once for every major integration time step.
 * Discrete states are typically updated here, but this function is useful
 * for performing any tasks that should only take place once per integration
 * step.
 */

static void mdlUpdate(x, u, S, tid)
   double *x;
   double *u;
   SimStruct *S;
   int tid;
{
}

/*
 * mdlDerivatives - compute the derivatives
 * This functionis used to compute the S-function block's derivatives.
 * The derivatives are placed in the dx variable.
 */

static void mdlDerivatives(dx, x, u, S, tid)
   double *dx;
   double *x;
```

```
   double *u;
   SimStruct *S;
   int tid;
{
            int v;
            v=u[4]+alpha*(u[3]-u[0]);
            if (u[1]<0.001)
            {
            u[1] = 0.001;
            dx[0] =-alpha*(x[0])+((N1-x[1])*(-u[0])+(N2-x[2])*(-u[0]*(u[1]*u[1]))+ (u[2]-x[6])*(-1)+
                        (N4-x[4])*(u[1]*((-(-x[1]*u[0]-x[2]*u[0]*(u[1]*u[1])-x[6])+v)/(x[4]*u[1])));
            dx[1] =(x[0])*(-u[0]);
            dx[2] =(x[0])*(-u[0]*(u[1]*u[1]));
            dx[3] =0;
            dx[4] =(x[0])*(u[1])*(-(-x[1]*u[0]-x[2]*u[0]*(u[1]*u[1])-x[6])+v)/(x[4]*u[1]);
            dx[5] =0;
            dx[6] =x[0]*(-1);
            }
            else
            {
            dx[0] =-alpha*(x[0])+((N1-x[1])*(-u[0])+(N2-x[2])*(-u[0]*(u[1]*u[1]))+ (u[2]-x[6])*(-1)+
                        (N4-x[4])*(u[1]*((-(-x[1]*u[0]-x[2]*u[0]*(u[1]*u[1])-x[6])+v)/(x[4]*u[1])));
            dx[1] =(x[0])*(-u[0]);
            dx[2] =(x[0])*(-u[0]*(u[1]*u[1]));
            dx[3] =0;
            dx[4] =(x[0])*(u[1])*(-(-x[1]*u[0]-x[2]*u[0]*(u[1]*u[1])-x[6])+v)/(x[4]*u[1]);
            dx[5] =0;
            dx[6] =x[0]*(-1);
            }
}

/*
 * mdlTerminate - called when the simulation is terminated.
*/

static void mdlTerminate(S)
   SimStruct *S;
{
}

#ifdef      MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Entrada Analógica-Digital

```
/*
 * Labpc_ad.c
 *
 *   S-Function device driver for the Computer Boards
 *   LabPC+ analog input section of the board.
 *
```

```
 */

#undef S_FUNCTION_NAME
#define S_FUNCTION_NAME        labpc_ad

#include <stddef.h> /* For NULL  */
#include <stdlib.h> /* For min   */

#include "simstruc.h" /* Where simulation structure, S, is defined */

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif

#ifndef min
#define min(x, y)    ((x) < (y) ? (x) : (y))
#endif

/* compiler dependant low level hardware calls */
#ifdef __HIGHC__
        #include <conio.h>
        #define    hw_outportb( portid, value )        _outp( portid, value )
        #define    hw_inportb( portid )                _inp( portid )
        #define    hw_outport( portid, value )         _outpw( portid, value )
        #define    hw_inport( portid )          _inpw( portid )
#elif __WATCOMC__
        #include <conio.h>
        #define    hw_outportb( portid, value )        outp( portid, value )
        #define    hw_inportb( portid )         inp( portid )
        #define    hw_outport( portid, value )         outpw( portid, value )
        #define    hw_inport( portid )          inpw( portid )
#else
static void
hw_outportb(portid, value)
int portid;
int value;
{
}

static int
hw_inportb(portid)
int portid;
{
}

static void
hw_outport(portid, value)
int portid;
int value;
{
}

static int
```

```
hw_inport(portid)
int portid;
{
}
#endif

/* Input Arguments */
#define BASE_ADDRESS_ARG                ssGetArg(S,0)
#define NUM_OF_CHANNELS_ARG             ssGetArg(S,1)
#define SAMPLE_TIME_ARG                 ssGetArg(S,2)
#define ACCESS_HW_ARG                   ssGetArg(S,3)
#define NUMBER_OF_ARGS                  (4)

#define NSAMPLE_TIMES           (1)

/* Storage Allocation */
#define BASE_ADDRESS        (0)
#define NUM_CHANNELS        (1)
#define ACCESS_HW           (2)
#define VOLTS_BIT           (3)

#define ADC_GAIN            (1)

/* Integer storage is used for above necessary storage.*/

#define NUMBER_OF_IWORKS    (4)

/* hardware registers */
#define COMR1(base_addr) (base_addr)                /* WRITE 0x00 FOR INIT BOARD*/
#define COMR2(base_addr) ((base_addr) + 0x01)       /* WRITE 0x00 FOR INIT BOARD*/
#define COMR3(base_addr) ((base_addr) + 0x02)       /* WRITE 0x00 FOR INIT BOARD*/
#define COMR4(base_addr) ((base_addr) + 0x0f)       /* WRITE 0x00 FOR INIT BOARD*/
#define CNTMA(base_addr)        ((base_addr) + 0x17)        /* WRITE 0x34 FOR INIT BOARD*/
#define CNTDA0(base_addr)       ((base_addr) + 0x14)        /* WRITE 0x0A AND 0x00FOR INIT
BOARD*/
#define DMATC(base_addr)        ((base_addr) + 0x0a)        /* WRITE 0x00 FOR INIT BOARD*/
#define TICR(base_addr)         ((base_addr) + 0x0c)        /* WRITE 0x00 FOR INIT BOARD*/
#define ADCLR(base_addr)  ((base_addr) + 0x08)      /* WRITE 0x00 FOR INIT BOARD*/
#define ADFIFO(base_addr)((base_addr) + 0x0a)       /* READ TWICE FOR INIT BOARD*/
#define DAC0L(base_addr)  ((base_addr) + 0x04)      /* WRITE 0x00 FOR INIT BOARD*/
#define DAC0H(base_addr)  ((base_addr) + 0x05)      /* WRITE 0x00 FOR INIT BOARD*/
#define DAC1L(base_addr)  ((base_addr) + 0x06)      /* WRITE 0x00 FOR INIT BOARD*/
#define DAC1H(base_addr)  ((base_addr) + 0x07)      /* WRITE 0x00 FOR INIT BOARD*/
#define STARTR(base_addr)       ((base_addr) + 0x03)
#define CNTDA1(base_addr)       ((base_addr) + 0x15)

#define STATUS(base_addr)       ((base_addr) + 0x00)

/* Status register bit definitions */
#define CONV_DONE       (0x01)
#define MASK            (0x80)

#define ADC_RESOLUTION      (4096)
```

```
/*
* Sample interval byte definition
* SINT=500 (0x01f4) for 0.5 ms of sampling interval (500  * 1usec)
*/
#define SINTL                   (0xf4)
#define SINTH                   (0x01)

/*
* Sample count byte definition
* SINT=3 (0x0003) for 3 sampling count
*/
#define SCNTL                   (0x03)
#define SCNTH                   (0x00)

static void mdlInitializeSizes(S)
        SimStruct *S;
{
        int num_channels;

        if (ssGetNumArgs(S) != NUMBER_OF_ARGS)
        {
#ifdef MATLAB_MEX_FILE
                mexErrMsgTxt("Wrong number of input arguments passed.\nfour arguments are
                expected\n");
#endif
        }
        /* Check the size of the number of channels,*/
        else
        {
                num_channels = mxGetPr(NUM_OF_CHANNELS_ARG)[0];
#ifdef MATLAB_MEX_FILE
                if ((num_channels < 1) || (num_channels > 8)) {
                        mexErrMsgTxt("Number of input channels must be between 1 and 8\n");
                }
#endif
                /* Set-up size information */
                ssSetNumContStates(S, 0);
                ssSetNumDiscStates(S, 0);
                ssSetNumOutputs(S, num_channels);
                ssSetNumInputs(S, 0);
                ssSetDirectFeedThrough(S, 0); /* No direct feed-through */
                ssSetNumSampleTimes(S, NSAMPLE_TIMES);
                ssSetNumInputArgs(S, NUMBER_OF_ARGS);
                ssSetNumIWork(S, NUMBER_OF_IWORKS);
        }
}

/* Function to initialize sample times */
static void mdlInitializeSampleTimes(S)
        SimStruct *S;
{
        ssSetSampleTimeEvent(S, 0, mxGetPr(SAMPLE_TIME_ARG)[0]);
```

```
        ssSetOffsetTimeEvent(S, 0, 0.0);
}

static void mdlInitializeConditions(x0, S)
        double *x0;
        SimStruct *S;
{
        int arg_str_len = 128;
        char arg_str[128];
        unsigned int base_addr;
        int num_channels;
        double volts_per_bit;
        int access_hw;
/*      unsigned int num_inputs;*/

        mxGetString(BASE_ADDRESS_ARG, arg_str, arg_str_len);
        base_addr = (unsigned long)strtol(arg_str, NULL, 0);
        ssSetIWorkValue(S, BASE_ADDRESS, (int)base_addr);

        num_channels = min(8, (int)mxGetPr(NUM_OF_CHANNELS_ARG)[0]);
        ssSetIWorkValue(S, NUM_CHANNELS, num_channels);


   /* Now save the resolution to integer storage space*/
        volts_per_bit = 10.0 / (ADC_GAIN * ADC_RESOLUTION);
        ssSetRWorkValue(S, VOLTS_BIT, volts_per_bit);


        access_hw = (int)(mxGetPr(ACCESS_HW_ARG)[0]);
        ssSetIWorkValue(S, ACCESS_HW, access_hw);

        /* If this is a MEX-file, then accessing the hardware
         * depends on the value of the access hardware flag in the
         * S-function's dialog box.
         */
#ifdef MATLAB_MEX_FILE
        if (!access_hw)
                return;
#endif  /* MATLAB_MEX_FILE */

        /* Init the Board */

        hw_outportb(COMR1(base_addr), 0x00);
        hw_outportb(COMR2(base_addr), 0x00);
        hw_outportb(COMR3(base_addr), 0x00);
        hw_outportb(COMR4(base_addr), 0x00);
        hw_outportb(CNTMA(base_addr), 0x34);
        hw_outportb(CNTDA0(base_addr), 0x0a);
        hw_outportb(CNTDA0(base_addr), 0x00);
        hw_outportb(DMATC(base_addr), 0x00);
        hw_outportb(TICR(base_addr), 0x00);
        hw_outportb(ADCLR(base_addr), 0x00);
        hw_inportb(ADFIFO(base_addr));
```

```
        hw_inportb(ADFIFO(bas e_addr));
        hw_outportb(DAC0L(base_addr), 0x00);
        hw_outportb(DAC0H(base_addr), 0x00);
        hw_outportb(DAC1L(base_addr), 0x00);
        hw_outportb(DAC1H(base_addr), 0x00);
}

/* Function to compute outputs */
static void mdlOutputs(y, x, u, S, tid)
        double *y;
        double *x;
        double *u;
        SimStruct *S;
        int tid;

{
        unsigned int base_addr = ssGetIWorkValue(S, BASE_ADDRESS);
        int num_channels = ssGetIWorkValue(S, NUM_CHANNELS);
        double volts_per_bit;
        unsigned int low_byte;
        unsigned int high_byte;
        unsigned char ADLow;
        unsigned char ADHigh;
        int i;
        int j;
        unsigned int num_inputs;


        /* If this is a MEX-file, then accessing the hardware
         * depends on the value of the access hardware flag in the
         * S-function's dialog box.
         */
#ifdef MATLAB_MEX_FILE
        int access_hw = ssGetIWorkValue(S, ACCESS_HW);
        if (!access_hw)
        {
                /* Allow the inputs to be directly passed to the
                 * outputs for a loopback situtation.  Note that
                 * to use this capability, an input port needs to
                 * be added to the block.
                 */
                for (i = 0; i < num_channels; i++) {
                        y[i] = u[i];
                }
                return;
        }
#endif  /* MATLAB_MEX_FILE */

        /*Set up the board for multiple A/D conversions with channels scanning*/

        /*Gain 1, straigth binary and channel scanning*/

        num_inputs = (unsigned int)(num_channels-1);
```

```
		num_inputs &= 0x07;
		hw_outportb(COMR1(base_addr), num_inputs);
		num_inputs |= MASK;
		hw_outportb(COMR1(base_addr), num_inputs);

		/*Program the sampling interval counter*/

		hw_outportb(CNTMA(base_addr), 0x34);
		hw_outportb(CNTDA0(base_addr), SINTL);
		hw_outportb(CNTDA0(base_addr), SINTH);

		/*Program the sample counter*/

		hw_outportb(CNTMA(base_addr), 0x70);
		hw_outportb(CNTDA1(base_addr), SCNTL);
		hw_outportb(CNTDA1(base_addr), SCNTH);

		/*Clear the A/D circuitry*/
		hw_outportb(ADCLR(base_addr), 0x00);
		hw_inportb(ADFIFO(base_addr));
		hw_inportb(ADFIFO(base_addr));

		/*Disable signal EXTCONV and select NRSE input configuration */

		hw_outportb(COMR4(base_addr), 0x10);

		/* Start conversion, set SWTRIG=1 (bit 2 of command register 2*/

		hw_outportb(COMR2(base_addr), 0x04);

		/* Wait for the A/D to finish its conversion,
		 * read from the two A/D addresses and store
		 * them in a double.
		 */

		for (j=0; j<num_channels; ++j)
		{
			while (~(hw_inportb(STATUS(base_addr))) & CONV_DONE);

			ADLow = hw_inportb(ADFIFO(base_addr));
			ADHigh = hw_inportb(ADFIFO(base_addr));
			low_byte = ADLow & 0xff;
			high_byte = (ADHigh << 8) &0xff00;
			volts_per_bit = ssGetRWorkValue(S, VOLTS_BIT);
			i=(num_channels-1)-j;
			y[i] = (double)((low_byte | high_byte) * volts_per_bit);
		}
}

/* Function to compute model update */
static void mdlUpdate(x, u, S, tid)
		double *x;
		double *u;
```

```
        SimStruct *S;
        int tid;
{
}

/* Function to compute derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
        double *dx;
        double *x;
        double *u;
        SimStruct *S;
        int tid;
{
}

/* Function to perform housekeeping at execution termination */
static void mdlTerminate(S)
        SimStruct *S;
{
}

#ifdef MATLAB_MEX_FILE          /* Is this file being compiled as a MEX-file? */
#include "simulink.c"           /* Mex glue */
#else
#include "cg_sfun.h"            /* Code generation glue */
#endif
```

Salida Digital-Analógica

```
/*
 * Labpc_da.c
 *
 *   S-Function device driver for the Computer Boards
 *   LabPC+ analog output section of the board.
 *
 */

#undef S_FUNCTION_NAME
#define S_FUNCTION_NAME       labpc_da

#include <stddef.h> /* For NULL  */
#include <stdlib.h> /* For min   */

#include "simstruc.h" /* Where simulation structure, S, is defined */

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#endif

#ifndef min
#define min(x, y)    ((x) < (y) ? (x) : (y))
#endif

/* compiler dependant low level hardware calls */
```

```
#ifdef __HIGHC__
        #include <conio.h>
        #define    hw_outportb( portid, value )        _outp( portid, value )
        #define    hw_inportb( portid )                _inp( portid )
        #define    hw_outport( portid, value )         _outpw( portid, value )
        #define    hw_inport( portid )                 _inpw( portid )
#elif __WATCOMC__
        #include <conio.h>
        #define    hw_outportb( portid, value )        outp( portid, value )
        #define    hw_inportb( portid )                inp( portid )
        #define    hw_outport( portid, value )         outpw( portid, value )
        #define    hw_inport( portid )          inpw( portid )
#else
static void
hw_outportb(portid, value)
int portid;
int value;
{
}

static int
hw_inportb(portid)
int portid;
{
        return(0);
}

static void
hw_outport(portid, value)
int portid;
int value;
{
}

static int
hw_inport(portid)
int portid;
{
        return(0);
}
#endif

/* Input Arguments */
#define BASE_ADDRESS_ARG                    ssGetArg(S,0)
#define NUM_CHANNELS_ARG                    ssGetArg(S,1)
#define SAMPLE_TIME_ARG                     ssGetArg(S,2)
#define ACCESS_HW_ARG                       ssGetArg(S,3)
#define NUMBER_OF_ARGS                      (4)

#define NSAMPLE_TIMES                       (1)

/* Storage Allocation */
#define BASE_ADDRESS            (0)
```

```
#define NUM_CHANNELS            (1)
#define ACCESS_HW                       (2)
#define NUMBER_OF_IWORKS                (3)


/* hardware registers */
#define COMR1(base_addr) (base_addr)                    /* WRITE 0x00 FOR INIT BOARD*/
#define COMR2(base_addr) ((base_addr) + 0x01)           /* WRITE 0x00 FOR INIT BOARD*/
#define COMR3(base_addr) ((base_addr) + 0x02)           /* WRITE 0x00 FOR INIT BOARD*/
#define COMR4(base_addr) ((base_addr) + 0x0f)           /* WRITE 0x00 FOR INIT BOARD*/
#define CNTMA(base_addr)            ((base_addr) + 0x17)        /* WRITE 0x34 FOR INIT BOARD*/
#define CNTDA0(base_addr)           ((base_addr) + 0x14)        /* WRITE 0x0A AND 0x00FOR INIT
BOARD*/
#define DMATC(base_addr)            ((base_addr) + 0x0a)        /* WRITE 0x00 FOR INIT BOARD*/
#define TICR(base_addr)             ((base_addr) + 0x0c)        /* WRITE 0x00 FOR INIT BOARD*/
#define ADCLR(base_addr)  ((base_addr) + 0x08)          /* WRITE 0x00 FOR INIT BOARD*/
#define ADFIFO(base_addr)((base_addr) + 0x0a)           /* READ TWICE FOR INIT BOARD*/
#define DAC0L(base_addr)  ((base_addr) + 0x04)          /* WRITE 0x00 FOR INIT BOARD*/
#define DAC0H(base_addr)  ((base_addr) + 0x05)          /* WRITE 0x00 FOR INIT BOARD*/
#define DAC1L(base_addr)  ((base_addr) + 0x06)          /* WRITE 0x00 FOR INIT BOARD*/
#define DAC1H(base_addr)  ((base_addr) + 0x07)          /* WRITE 0x00 FOR INIT BOARD*/
#define STARTR(base_addr)           ((base_addr) + 0x03)
#define STATUS(base_addr)           ((base_addr) + 0x00)

#define DAC_RESOLUTION          (4095)
#define DAC_MAX_VOLTAGE         (4095)
#define DAC_MIN_VOLTAGE         (0)

static void mdlInitializeSizes(S)
        SimStruct *S;
{
        int num_channels;

        if (ssGetNumArgs(S) != NUMBER_OF_ARGS) {
#ifdef MATLAB_MEX_FILE
                mexErrMsgTxt("Wrong number of input arguments passed.\nFive arguments are
                expected\n");
#endif
        }
        /* Check the size of the number of channels*/
        else {
                num_channels = mxGetPr(NUM_CHANNELS_ARG)[0];
#ifdef MATLAB_MEX_FILE
                if ((num_channels < 1) || (num_channels > 2)) {
                        mexErrMsgTxt("Number of output channels must be between 1 and 2\n");
                }
#endif
                /* Set-up size information */
                ssSetNumContStates(S, 0);
                ssSetNumDiscStates(S, 0);
                ssSetNumOutputs(S, 0);
                ssSetNumInputs(S, 1); /*consider only one channel*/
                ssSetDirectFeedThrough(S, 1); /* Direct dependency on inputs */
```

```
                ssSetNumSampleTimes(S, NSAMPLE_TIMES);
                ssSetNumInputArgs(S, NUMBER_OF_ARGS);
                ssSetNumIWork(S, NUMBER_OF_IWORKS);

        }
}

/* Function to initialize sample times */
static void mdlInitializeSampleTimes(S)
        SimStruct *S;
{
        ssSetSampleTimeEvent(S, 0, mxGetPr(SAMPLE_TIME_ARG)[0]);
        ssSetOffsetTimeEvent(S, 0, 0);
}

static void mdlInitializeConditions(x0, S)
        double *x0;
        SimStruct *S;
{
        int arg_str_len = 128;
        char arg_str[128];
        unsigned int base_addr;
        int num_channels;
        int access_hw;

        mxGetString(BASE_ADDRESS_ARG, arg_str, arg_str_len);
        base_addr = (unsigned long)strtol(arg_str, NULL, 0);
        ssSetIWorkValue(S, BASE_ADDRESS, (int)base_addr);

        num_channels = min(2, (int)mxGetPr(NUM_CHANNELS_ARG)[0]);
        ssSetIWorkValue(S, NUM_CHANNELS, num_channels);


        access_hw = (int)(mxGetPr(ACCESS_HW_ARG)[0]);
        ssSetIWorkValue(S, ACCESS_HW, access_hw);

        /* If this is a MEX-file, then accessing the hardware
         * depends on the value of the access hardware flag in the
         * S-function's dialog box.
         */
#ifdef MATLAB_M EX_FILE
        if (!access_hw)
                return;
#endif  /* MATLAB_MEX_FILE */

        /* Init the Board */
/*
* Initialize the board outputs to be zero volts (0)
*/
        hw_outportb(DAC0L(base_addr), 0x00);
        hw_outportb(DAC0H(base_addr), 0x00);
        hw_outportb(DAC1L(base_addr), 0x00);
        hw_outportb(DAC1H(base_addr), 0x00);
```

```
}

/* Function to compute outputs */
static void mdlOutputs(y, x, u, S, tid)
	double *y;
	double *x;
	double *u;
	SimStruct *S;
	int tid;
{
	/* The voltage value is converted to binary, broken into one byte
	 * and one nibble and written out to the DAC.
	 */
	unsigned int base_addr = ssGetIWorkValue(S, BASE_ADDRESS);
	int num_channels = ssGetIWorkValue(S, NUM_CHANNELS);

	int low_byte;
	int high_byte;
	int i;
	int value;

	/* If this is a MEX-file, then accessing the hardware
	 * depends on the value of the access hardware flag in the
	 * S-function's dialog box.
	 */
#ifdef MATLAB_MEX_FILE
	int access_hw = ssGetIWorkValue(S, ACCESS_HW);
	if (!access_hw)
	{
		/* Allow the inputs to be directly passed to the
		 * outputs for a loopback situtation.  Note that
		 * to use this capability, an output port needs to
		 * be added to the block.
		 */
		for (i = 0; i < num_channels; i++) {
			y[i] = u[i];
		}
		return;
	}
#endif  /* MATLAB_MEX_FILE */

		value = (int)(DAC_RESOLUTION* (u[0] / 10.0));
		if (value > DAC_MAX_VOLTAGE) {
			value = DAC_MAX_VOLTAGE;
		}
		if (value < DAC_MIN_VOLTAGE) {
			value = DAC_MIN_VOLTAGE;
		}

		low_byte = (value) & 0xff;
		high_byte = (value >> 8) & 0xff;

/*write to DAC port*/
```

```
                i = num_channels;
                switch (i) {
                        case 1:
                                hw_outportb(DAC0L(base_addr), low_byte);
                                hw_outportb(DAC0H(base_addr), high_byte);

                        break;
                        case 2:
                                hw_outportb(DAC1L(base_addr), low_byte);
                                hw_outportb(DAC1H(base_addr), high_byte);
                        break;
                }

}
/* Function to compute model update */
static void mdlUpdate(x, u, S, tid)
        double *x;
        double *u;
        SimStruct *S;
        int tid;
{
}

/* Function to compute derivatives */
static void mdlDerivatives(dx, x, u, S, tid)
        double *dx;
        double *x;
        double *u;
        SimStruct *S;
        int tid;
{
}

/* Function to perform housekeeping at execution termination */
static void mdlTerminate(S)
        SimStruct *S;
{
        /*
         * This function zeros the DAC outputs.
         */
        unsigned int base_addr = ssGetIWorkValue(S, BASE_ADDRESS);
        int num_channels = ssGetIWorkValue(S, NUM_CHANNELS);
        int i;

        /* If this is a MEX-file, then accessing the hardware
         * depends on the value of the access hardware flag in the
         * S-function's dialog box.
         */
#ifdef MATLAB_MEX_FILE
        int access_hw = ssGetIWorkValue(S, ACCESS_HW);
        if (!access_hw)
                return;
```

```
#endif  /* MATLAB_MEX_FILE */

                    i = num_channels;

                    switch (i) {
                            case 1:
                                    hw_outportb(DAC0L(base_addr), 0x00);
                                    hw_outportb(DAC0H(base_addr), 0x00);

                            break;
                            case 2:
                                    hw_outportb(DAC1L(base_addr), 0x00);
                                    hw_outportb(DAC1H(base_addr), 0x00);
                            break;
                    }
}

#ifdef MATLAB_MEX_FILE          /* Is this file being compiled as a MEX-file? */
#include "simulink.c"           /* Mex glue */
#else
#include "cg_sfun.h"            /* Code generation glue */
#endif
```